PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 5 :
G10H 1/00, 1/40

A1

(11) International Publication Number: WO 93/22762

(43) International Publication Date: 11 November 1993 (11.11.93)

(21) International Application Number: PCT/US93/03667

(22) International Filing Date: 20 April 1993 (20.04.93)

(30) Priority data:
07/874,354 24 April 1992 (24.04.92) US

(71) Applicant: THE WALT DISNEY COMPANY [US/US]; 500 South Buena Vista Street, Burbank, CA 91521 (US).

(72) Inventors: REDMANN, William, Gibbens ; 3152 Dalhart Avenue, Simi Valley, CA 93063 (US). PETERSON, Michael, Harvey ; 1265 Linden Avenue, Glendale, CA 91201 (US).

(74) Agent: SCHUYLER, Marc, P.; Pretty Schroeder Brueggemann & Clark, 444 South Flower Street, Suite 2000, Los Angeles, CA 90071 (US).

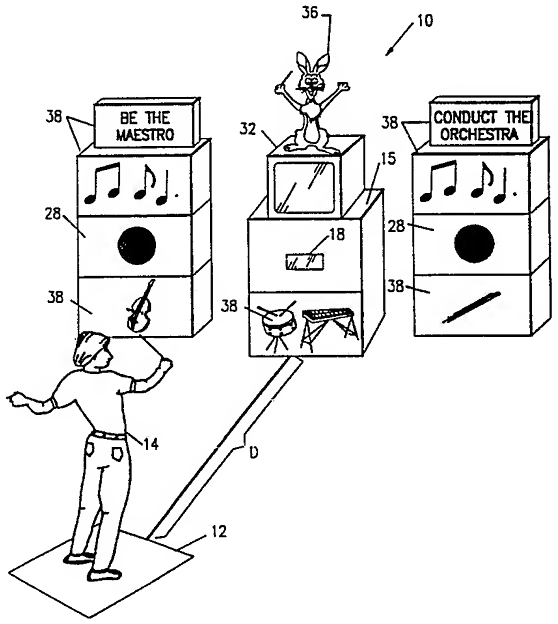
(81) Designated States: JP, European patent (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
With international search report.

(54) Title: APPARATUS AND METHOD FOR TRACKING MOVEMENT TO GENERATE A CONTROL SIGNAL

(57) Abstract

The invention permits the generation of multipurpose control signals by tracking movement that occurs within a field of view. In particular, a "Guest Controlled Orchestra" utilizing these inventive principles permits a layman guest to step into the shoes of an orchestra conductor, and through image processing, conduct the performance of a prerecorded music score. A video camera captures a field of view encompassing the guest for generation of a digital image. The field of view is sampled in left and right windows and the intensity of pixels within the windows are compared with a past image to determine if intensity change exceeds a predetermined threshold. A center of movement is computed for each window by averaging coordinates of each such pixel, and the centers of movement stored for future use. By analyzing change in centers of movement, tempo and volume are derived. Volume is derived from the quantity of pixels that correspond to the predetermined intensity change, and which therefore represent movement. Prerecorded audio data are formatted into MIDI audio commands, and together with video frame advance commands, are processed and output in response to these derived signals.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AT	Austria	FR	France	MR	Mauritania
AU	Australia	GA	Gabon	MW	Malawi
BB	Barbados	GB	United Kingdom	NL	Netherlands
BE	Belgium	GN	Guinea	NO	Norway
BF	Burkina Faso	GR	Greece	NZ	New Zealand
BG	Bulgaria	HU	Hungary	PL	Poland
BJ	Benin	IE	Ireland	PT	Portugal
BR	Brazil	IT	Italy	RO	Romania
CA	Canada	JP	Japan	RU	Russian Federation
CF	Central African Republic	KP	Democratic People's Republic of Korea	SD	Sudan
CG	Congo			SE	Sweden
CH	Switzerland	KR	Republic of Korea	SK	Slovak Republic
CI	Côte d'Ivoire	KZ	Kazakhstan	SN	Senegal
CM	Cameroon	LI	Liechtenstein	SU	Soviet Union
CS	Czechoslovakia	LK	Sri Lanka	TD	Chad
CZ	Czech Republic	LU	Luxembourg	TG	Togo
DE	Germany	MC	Monaco	UA	Ukraine
DK	Denmark	MG	Madagascar	US	United States of America
ES	Spain	ML	Mali	VN	Viet Nam
FI	Finland	MN	Mongolia		

APPARATUS AND METHOD FOR TRACKING MOVEMENT TO GENERATE A CONTROL SIGNAL

BACKGROUND

The present invention relates to an apparatus and method for generating a control signal in response to movement, and more particularly, relates to a music generator that extracts information for "conducting" music, in the same sense that a conductor would conduct an orchestra.

People enjoy music. Many especially enjoy classical music and appreciate the role of the conductor. Typically, a conductor will orchestrate music individually played by over one hundred independent instruments into one united harmonious score. The styles and products of different conductors are as unique as the music scores that they conduct.

The conductor is viewed as the head of the orchestra, its leader, and frequently, is the recipient of praise for his creativity and his ability to transform the work of a composer into a derived, artistically unique musical product.

Many have fantasized being a conductor and being able to create such a unique musical creation. However, for most, this fantasy will not be achieved, because of the difficulty in learning and in mastering the conductor's unique "language".

The conductor must be fluent in expressing conducting information, used to produce the final musical product, to each member of the orchestra. This information is transmitted through the gestures and movements of the conductor and is based upon his knowledge and experience in music in general, his knowledge of the music score which is to be performed, his own style and taste, and the knowledge and experience and style of each member of the orchestra. Learning the conductor's musical experience and the other information necessary to conduct an orchestra is fairly difficult on its own, but in addition, the layman must also learn the language by which a conductor communicates his expression and taste to enable the members of the orchestra to play in unison at the correct tempo, volume, emphasis and presence.

Thus, the conductor's musical skills necessary to synchronize and direct the playing of music typically exceed those of the common person. However, this inability does not eliminate the desire that many have to express their own musical taste and style. To this end, equipment and methods have been developed over recent years which attempt to enable a common person to step into the shoes of the conductor, and to create a unique musical product, stylized by their own expression.

One such system, employed for many years at "EPCOT Center," at Walt Disney World, Florida, enables one to simulate the role of conductor by mixing instrument tracks that correspond to a prerecorded musical score. Using this system, one assumes a defined spot and moves and gestures as if he or she were the conductor. Four sonar devices are used to each derive the relative distance of a hand intersecting a sonar beam to a background object normally struck by the beam. This relative distance is then used to raise or lower corresponding tracks of a prerecorded musical score.

Another method utilizes video processing to isolate the outline of the form of a guest conductor against a distinguished background and to derive information from the orientation of the person's outline. According to this method, the repeatedly captured outline can be analyzed for movement, and direction or speed of movement determined from recognized states can be extracted and used or analyzed to modify sound.

Still another system attaches motion sensors to a guest conductor. These sensors, which may include, for example, motion sensing gloves or the like, sense acceleration or movement relative to other sensors, and thereby provide an electronic signal that can be used to generate music.

Each of these systems has disadvantages that offer room for improvement and modification. For example, in the video system mentioned above, the guest conductor and a background must be specifically contrasted in order to allow the video equipment to provide a stark contrast to capture the guest conductor's outline.

Other difficulties are also present in a video device. Video systems generally use image processing equipment that transforms the video signal into "pixels", i.e., numbers that each represent sampled visual characteristics at distinct locations scanned by a video camera. These video systems produce many thousands of "pixels" per second. Typically, the known methods of processing these pixels, such as by tracing only the outline, must rely on some shortcut in order to track movement. The huge number of pixels and complex processing that is required make full image processing a practical impossibility for real time applications.

With respect to the sonar system mentioned above, its field of view is extremely limited since the sonar waves are either directed to, or are received from, a specific point in order to accurately locate the guest conductor's hands and to distinguish them against other sound reflecting backgrounds. Furthermore, the nature of the sonar system dictates that only a limited range of guest activities will produce results. This limitation in the range of activities detracts from the guest's freedom of expression and makes it more difficult for a guest to create music by imitating his or her mental impression of a conductor in action.

Accordingly, there has existed a definite need for an apparatus or method which can generate a control signal in response to movement and use that signal to alter the performance parameters of prerecorded music. Such a system would need to track movements which occur within a field of view. Additionally, it should provide a method for processing that employs a reference back to specific, previously identified pixels derived from a video image to enable a digital processing system to process the movement in real-time. This needed apparatus or method should be applicable in particular to a device that permits the controlled performance of music in response to tracked conductor parameters.

The present invention fulfills these needs, overcomes many of the aforementioned disadvantages and provides an improved and unique apparatus and method for playing music and for allowing a guest conductor to direct a musical score. In broader terms, however, the present invention provides a unique and novel apparatus and method for generating control signals which may be applied to a wide variety of tasks by tracking movements that occur within a field of view.

SUMMARY OF THE INVENTION

The present invention provides a novel control system that generates a control signal by tracking movement within a field of view, and which can track gestures and movements of a person occupying that field of view. In this manner, movement may be utilized in a wide variety of applications to orchestrate system control through derived electronic signals. In particular, the specific contemplated application of the invention is to a Guest Controlled Orchestra that permits a guest to step into the shoes of an orchestra conductor and direct the performance of music that tracks his movements, much as a real orchestra would a professional conductor.

The invention provides a digital motion processing system having an imaging device that repeatedly captures a field of view, which is then processed by image and data processing elements to yield relative movement between frames. From the comparison of the two images, the processing elements determine a single value representative of positions of the current image that correspond to movement. In response to this single value a signal generator generates at least one control signal. Since the preferred embodiment is a music system, the signal generator may be included in a sound generator that uses the generated control signals to generate and format sound.

The method of processing information from a field of view to generate this control signal includes generating and storing a first image representing the field of view at a first point in time, and generating a second image representing the field of view at a second different point in time. Pixels from these two images are compared to determine a set of pixels that represent movement of the second image relative to the first image. From the locations of those pixels in the set that represent movement, a single value is computed. The control signal is generated in response to this single value.

More particularly, the apparatus utilizes an imaging device to provide an electronic signal representative of a field of view captured by the imaging system. This electronic signal, representing a sequence of scans of the field of view, is digested by an image digitizer that processes the electronic signal to yield a sequence of numbers. Each number corresponds to the intensity of a particular point in the field of view and thus represents a specific, addressable location, or "pixel", within that field of view. This numeric data is then processed in a series of steps to efficiently permit the tracking of movement within the field of view.

The intensity of pixels of an image are compared with the intensity of a previous image at the same pixel location to determine those pixels that represent movement relative to the previous image. A difference in corresponding pixel values that exceeds a selected threshold is taken as an indication of motion. The data processor computes a single value representative of all the pixels that represent movement for a given image.

At a more specific level of the invention, the data processor computes a single position, or "centroid", for each "window" or predefined subportion of the field of view which is to be specifically analyzed for movement. It then computes, from all such single positions, the single value, or "scalar", which it then analyzes to generate the control signal. For example, in the case of the preferred embodiment described below, x and y indices of two such centroids for each image are simply summed to yield a single number, or "scalar", representing movement within the field of view.

Following a more particular form of the invention, after processing several image frames, the relative magnitude of several of the single values representing recent frames is correlated over a larger group of the single values to determine how fast or slow prerecorded music should be played. By recognizing specific guest actions, more particular forms of the invention also provide for selection of different instruments which can be alternatively used to play the same notes, and for volume control. The latter is dependent upon the number of pixels for a given image frame that represent movement.

Another feature of the motion processing system in accordance with the invention provides for control of a video display in response to tracked movement. For example, a prerecorded video image, such as a cartoon, may be visually displayed to the user or others at a rate that tracks the user's movements occurring within the field of view. This latter embodiment correctly emphasizes the current invention as providing a method of generation control signals in general, with many possible applications.

The invention may be better understood by referring to the following detailed description, which should be read in conjunction with the accompanying drawings. The detailed description of a particular preferred embodiment, set out below to enable one to build and use an example of the invention, are not intended to limit the claims but to serve as a particular example thereof.

DESCRIPTION OF THE DRAWINGS

FIG. 1 is a perspective view of an electronic guest controlled orchestra system in accordance with the invention;

FIG. 2 is a perspective view of the system illustrated as in FIG. 1, with the components of a console shown in greater detail;

FIG. 3 is a flow chart illustrating operation the main processing functions of a data processor used in the guest conducted orchestra system shown in FIG. 1;

FIG. 4 is a flow chart of the pixel sampling function, referred to in FIG. 3, that illustrates in greater detail the steps used to accomplish pixel sampling and centroid computation;

FIG. 5 is a flow chart of a background schedule filling process for the MIB that operates in parallel with the main processing of the data processor and that controls the loading of MIDI note commands from a computer to the MIB used in the guest conducted orchestra system of FIG. 1;

FIG. 6 is a flow chart which illustrates in greater detail a correlation process referred to in the flow chart shown in FIG. 3;

FIG. 7 is a flow chart which illustrates maxima detection in accordance with FIG. 6; and,

FIG. 8 is a flow chart which illustrates minima detection in accordance with FIG. 6.

DETAILED DESCRIPTION

In accordance with the principles of the invention which have been summarized in the section above, the inventors have developed a preferred implementation of their invention which will be further described below. This preferred embodiment is called a "Guest Controlled Orchestra", and allows a guest to imitate the actions of a conductor, and to observe how those actions affect the generation of music and the display of animation. The "orchestra" may be an ensemble of at least one orchestral voice, which may be an instrument or voice, or any other means of generating sound. Thus, the preferred embodiment may be applied to allow the guest to conduct nearly any ensemble, from a single instrument to a choir, to a rock group, etc. However, it is emphasized that the invention, as described above, relates not just to a music system, but is more broadly a system for generating control signals from movement.

The preferred use of the Guest Controlled Orchestra is in a theme park, and for that reason, it has been designed with smooth and simple guest interaction and throughput in mind. Pursuant to these criteria, the system has been designed to provide intuitive operation to a guest, and so does not require an employee/operator or specific instruction before use.

Referring first to FIGS. 1 and 2, the Guest Controlled Orchestra 10 includes a conductor station 12, on which a guest 14 may stand to perform conducting movements with his or her arms, and a console 15 spaced a predetermined distance D from the conducting station and facing toward the person. The console 15 conceals from the person a monitoring means for optically scanning movements of the guest and which is aligned towards the conductor station for this purpose and a computer processing means for digesting the scanned movements of the guest and generating and applying control signals. The console 15 also conceals a music storage means for storing electronic digital information that represents a musical piece to be played and audio playing means for playing the musical piece in response to the electronic digital information, as modified by the computer processing means.

More particularly, the system 10 includes a video camera 16, aligned to monitor the guest 14 through an infrared window 18, a computer 20 for performing image and data processing tasks, a MIDI synthesizer 22, which converts MIDI format music information supplied by the computer into a signal for mixing by a mixer 24 and audio amplification by an amplifier 26, and a speaker system 28 for ultimate rendition of a prerecorded musical score that has been varied in accordance with the guest's conducting actions. The computer 20 is fitted with an image digitizer add-on board for digitizing the camera's analog video output into a form that can be read and processed by the computer. In addition, the computer is also fitted with a musical/personal computer interface add-on board that controls the timing of music commands for the MIDI synthesizer. A video disk player 30 and monitor 32 are coupled to the computer to permit display of animation that is synchronized to the guest's tempo, as well as an additional monitor 34 that displays a special viewing image that represents detection of movement by the system 10.

The system 10 also includes a conductor statuette 36 and appropriate suggestive materials, illustrated in FIG. 1 as signs and notes 38, to indicate to the guest to imitate the actions of a conductor in order to achieve the desired result of generating music. It has been found that such suggestive materials are necessary and sufficient to cause the guest to intuitively operate the Guest Controlled Orchestra.

Within the conductor station 12, the guest occupies a field of view which is scanned by the video camera 16. The preferred embodiment captures this field of view and analyzes "windows" where independent movement is expected, such as one window about the left arm, and one window about the right arm. Through image processing, the guest's image is separately processed for each window so that movement of each arm yields control information used to process instrument, volume and tempo information that will influence the rendition of the prerecorded music score. The field of view may optionally be divided into as many windows as necessary that will yield distinct physical movement, such as movement of a finger, for example. These "windows" may be dynamic, or made to change in position relative to the field of view, to track objects and movement. In the Guest Controlled Orchestra, however, only two static windows are used to separately process movement of the left and right arms.

Each window may be processed to yield one or more control signals. Alternatively, a single control signal may be the product of several independent windows. The Guest Controlled Orchestra yields four control signals which influence the playing of the musical score. These signals include tempo, left only volume, right only volume and volume for instruments to be applied to both speakers.

With reference to FIG. 2, the contents of the console 15 are shown as removed from their normal position of concealment from the guest. The video camera 16 repeatedly scans the field of view and thereby captures the guest's image through the infrared window 18. The computer 20 is coupled to the video camera 16, so as to receive its thirty frame-per-second input and process it to generate MIDI format music commands, as well as commands to control the playing of a video disk. The console 15 also conceals the professional style video disk player 30, which receives these commands, i.e., 1-5 frame advances, and directs the display of prerecorded animation stored on the video disk that also is sequenced to the guest's actions. The computer 20 processes audio data, which is stored internally in its RAM, to format MIDI commands and set volume and tempo information for the ultimate rendition by the sound system. The synthesizer 22 receives the MIDI output of the computer and generates an analog electronic signal which may be used to drive the speakers 28. Selection of the acoustic elements, namely, the MIDI synthesizer 22, mixer 24, audio amplifier 26 and speakers 28, is well within the skill of one familiar with electronic music equipment and hence, will not be discussed in detail.

The console 15 conceals the video disk player 30, which is a professional type video disk player that may be used both to record and play, and which is controlled by a control signal output by the computer. Video frames are output by the video disk player in response to the tempo information generated by the computer 20 and coupled to the disk player 30 via an RS-232 connector. The disk player's output feeds the video monitor 32 to display visual data stored on the disk player to the guest conductor 12 and observers. In the case of the preferred embodiment, the video monitor displays animation, including animated sea creatures, that appear to dance or otherwise perform in synchronization with the music and the guest conductor's actions. Derived control of video display is yet another example of the application of the invention to a system for generating control signals in general.

The processing hardware, including the image processor, the data processor and their supporting memory, is embodied in the personal computer 20, as modified with the addition of add-on boards, including an image digitizer and a MIDI/personal computer interface. More specifically, the video frame store chosen for the system 10 is an "IVG-128" board which is available from Datacube, Inc., of Peabody, Massachusetts. The MIDI/personal computer interface is analogous to a "MPU401" board, sold under the designation "MQX-32M", available from Music Quest, Inc., of Plano, Texas, and will hereafter be referred to as the musical interface board ("MIB"). The personal computer used in the preferred embodiment is an IBM 386-25 personal computer. Graphics and data processing software, written by the inventors in the "C" language and partly in machine language, directs the CPU's performance of the various tasks and the CPU's interface with these two add-on boards. The software, which is illustrated in the flow charts shown in FIGS. 3-8, is described in functional terms in the paragraphs that follow.

The video camera 16 chosen is a model "TI-24A", available from the NEC Corporation of Tokyo, Japan. It generates a black and white electronic output of standard video format, elaborated upon below. While the preferred embodiment uses a black and white camera, which will typically capture portions of the infrared spectrum, any type of camera may be used in accordance with the principles of the invention. The infrared window 18, discussed above, is implemented not for the desirability of capturing the infrared spectrum, but primarily to hide the video camera from observance by the guest and yet allow the camera to capture the field of view.

The video camera 16 generates a standard video signal by scanning the field of view and producing sixty interlaced fields per second, or thirty complete frames per second. An electronic signal is produced by the video camera which represents the monochromatic luminance of the field of view as it is line-scanned from left to right. Each scan is slightly offset vertically from other scans, such that five hundred and twenty-five horizontal lines of scanning are used to cover the entire field of view. The electronic video signal, which repeats itself for each frame or thirty times per second, represents a generally continuous trace of the picture of five hundred and twenty-five horizontal lines when placed adjacent to one another. More precisely, each frame is comprised of two interlaced fields of alternating lines: The camera scans twice for each frame, scanning every other line each time, and produces five hundred and twenty-five lines. By scanning a picture in discrete lines comprising interlaced fields, the video camera produces a video electronic signal from a picture in approximately the reverse manner that a television reproduces a picture from a video signal.

This video signal is coupled to the personal computer 20 for image and data processing. During receipt of the video signal from the video camera 16, the computer's "IVG-128" board digitizes the video signal and thereby generates a plurality of pixels that represent the video signal. A "pixel" is nothing more than a sample of the video signal from which luminance can be discerned for a particular location within the field of view. Since the system 10 is a digital system, each video signal is digitized to provide a number for each pixel. This number comprises 8 bits that identify a sampled monochromatic (black-and-white) luminance for that pixel.

Thus, the "IVG-128" board of the computer 20 converts the repeated electronic video signal from the imaging system into a sequence of numbers, each number corresponding to the luminance of the field of view at a particular point, or pixel. As will be discussed further below, each pixel corresponds to a location and is represented by coordinates that represent its location within the field of view.

Thus, as described above, the video camera produces an electronic signal that scans left to right across the field of view producing approximately 262-1/2 lines of visual information, or a single interlaced field, sixty times per second. In addition, the electronic signal from the video camera contains synchronization information that is utilized by equipment receiving the electronic signal to reconstitute the visual image within the field of view.

In digitizing the video signal, the "IVG-128" board produces two types of digital information in response to this electronic signal for processing by the computer. First, it produces 8 bit digital values that represent the sampled black-to-white luminance of a pixel, or of a particular position within the captured field of view. Second, it produces status flags, or bits for sampling by the computer, which indicate when the pixel generator has digitized each of the first and second interlaced fields of a given frame of visual data. It is necessary for the computer to monitor these status flags to wait for and synchronize the commencement of its main program loop with the digitizer's completion of the first interlaced field.

The visual pixel information is stored in four 64 k banks of random access memory (which can be looked at as a 512 by 512 single byte frame buffer), resident on the "IVG-128" board. Like any other type of random access memory ("RAM"), the pixel values may be overwritten with digital information to modify the stored image. The system 10 makes use of this ability in providing an output to the second monitor 34.

The computer's CPU compares a defined subset, i.e., every fourth pixel every sixth line, of these pixels with corresponding information of a previous image that has been stored in the computer's random access memory. In other words, the computer compares the pixel's number representative of luminance with a number representing luminance at the same coordinates from a previous image.

These sampling steps, described below, are described as including fixed increments. However, these constants are defined at the beginning of the program, and may be chosen in the discretion of the computer operator to be any practical value. These constants are described below as specific values, because it is these values which have been used in operation of the preferred system.

To sample and process the pixels, the computer first waits for the "IVG-128" to raise a status flag that indicates completion of the first of the two interlaced fields that make up each video frame. It is only this first field that is sampled by the computer's CPU for visual data. The CPU starts with column 25 (of 384 columns that contain visual data) and row 50 (of 485 rows that contain visual data) and reads every fourth pixel until forty pixels have been read. It then moves six lines below, i.e., column 25, row 56, and repeats this same procedure. When fifty rows of forty horizontal samples are read, i.e., a left window of the image corresponding to the location of the right arm, the CPU reads another 40 x 50 sample block, beginning with column 208, row 50, and proceeding every 4th pixel, every 6th row, to develop a right window of the image corresponding to the left arm.

The "IVG-128" board stores the digitized image in an address format with the least significant address bits containing the horizontal coordinates, i.e., 0-1FF Hex, or

0-511, and the most significant address bits, i.e., 200-3FE00 Hex, or multiples of 512, containing the vertical coordinates, or row information. This information is organized into four banks of 64k random access memory and requires selection of a particular bank in accordance with the "IVG-128"'s specifications and a standard sixteen bit address and CPU read and write operations. As indicated, although there are 512 possible columns or horizontal positions for each row, only 384 pixels contain visual information and only 80 of these are looked at by the CPU. Similarly, while the video signal has 525 interlaced lines per frame, only 485 contain visual information. It is therefore seen that by sampling only odd rows, the CPU scans only the first interlaced video field, and must complete its processing tasks before the completion of digitization of the next first interlaced field.

As the number representing each pixel is read by the CPU from the "IVG-128"'s memory, it is compared with a number representing the same pixel of a previous frame, saved to the computer's random access memory. After performing the comparison steps, described below, the computer writes the new number over the old number representing the previous frame, such that the new frame samples are stored in the computer's RAM and serve as the "previous frame" pixel data the next time the computer performs a comparison, one-thirtieth of a second later, for the next frame. It does this for each pixel that was used in the comparison regardless of magnitude of the luminance change, and thus requires 2 x 40 x 50 bytes of memory, or 4 k RAM, for the task.

In this manner, the computer ascertains which of the sampled subset of pixels represent movement relative to the earlier image. "Movement" is represented by a change in intensity, or luminance, as referred to above.

Luminance will change somewhat for most pixels of the captured image. It is therefore necessary for the computer to identify only those pixels for which the change in luminance is sufficiently great that the new luminance represents actual movement, as opposed to a slight change in lighting, or the like. To this end, as indicated in the software block diagram of FIG. 4, the CPU subtracts the luminance of a particular pixel from the old number corresponding to that same pixel, i.e., the same location within the field of view, from the immediately previous image. If the absolute value of the difference exceeds a predetermined number, the computer adds special x-y indices corresponding to that pixel to a x index sum and a y index sum of all pixels that likewise are sufficiently different from the previous image for that window. In the Guest Controlled Orchestra, the +/- difference between numbers is compared with both 10 hex and F0 hex, rather than the above-mentioned use of the absolute value of the difference, as computation of the absolute value is equivalent but requires additional software steps.

The computer writes each new pixel into the pixel sample buffer corresponding to each window, until all such pixels are exhausted. Therefore, in the Guest Controlled Orchestra, the computer 20 performs these steps for each sampled pixel of each of the left window and the right window. When it finishes each window, it will have a x index sum and a y index sum corresponding to that window.

In addition, when the computer 20 determines that a pixel has changed its luminance sufficiently from the past image, it increments a pixel count corresponding to the analyzed window. In other words, the computer begins the pixel count at zero each time it begins analyzing a window for movement. It then keeps track of the number of pixels of each window that represent movement. Each of the x index sum and y index sum corresponding to that window are subsequently divided by this "pixel count". The result is a single x and y index value that represents a "centroid", or single position within the window that is the average of all points which changed significantly in luminance, or which represent movement. As mentioned, centroids are computed for each window frame. As will be discussed further below, the system 10 uses the pixel counts for each window to also determine the audio volume instruments, or orchestral voices, of the music played.

The x and y index corresponding to each sampled pixel should not be confused with its video row (of 512) and column (of 512) coordinates which necessitate an 18-bit address. Rather, the CPU performs a simpler task of assigning a row number, commencing with 49 and decreasing to 0, and a column number, commencing with 39 and decreasing to 0, which it uses for its processing tasks.

As shown in FIG. 4 and the appended software listings, the CPU looks at the pixels in step wise fashion, as described above, in three stages. For each window, the CPU first searches for a valid pixel, comparing each new pixel sample with the corresponding old pixel sample until it detects a difference greater than 10 hex or less than F0 hex, writing each new pixel into the pixel sample buffer as it does so. When the CPU has found a first valid pixel, it samples 10 rows each stepped 6 lines apart, adding x and y index values of qualifying pixels to a corresponding cumulative sum and increasing the pixel count, as described above. Finally, after all ten rows have been sampled, the CPU ceases its comparison and testing functions and merely writes each remaining new pixel (in the 40 x 50 sample window) to the pixel sample buffer, overwriting the corresponding old pixel values as it does so. If at any time the CPU reaches the 2000th pixel for the window, i.e., row index = 0 and column index

= 0, it determines that it has reached the last sample, ceases all processing functions, and proceeds to the second window. As the CPU finishes each window, it computes a "centroid", or movement center point for that window. Again, although the sampling for the second window commences at column 208, line 50 for the second window, the CPU defines an index value of y = 49, x = 39, and decrements these coordinates as the second window is sampled, to compute a second centroid corresponding to the second window.

Thus, as the guest 14 waves each of his left and right arms, the computer 20 tracks these movements and generates a single point, thirty times per second for each of the left and right image windows, to represent movement relative to the immediately preceding frame. Ideal operation of the system makes some assumptions. The first is that the guest 14 remains at the conductor station 12 so that his movement is captured by the video camera. Second, it is theoretically possible for the guest to swing his arms across his body in such a manner that the sum of both centroids produces a constant sum, in which case the system will detect a minimum tempo. It is stressed, however, that the system as described is robust against the guest's arms crossing his body so that any motion, including body swaying, will enable detection of a tempo.

In the Guest Controlled Orchestra, the field of view, as digitized, is sampled in static left and right half windows, which capture the guest's movements as long as he remains in the conductor station 12. Thus, the same subset of pixels for each window are always used, and their values subsequently stored in computer's RAM, for use as the prior image in the subsequent comparison step for the next frame. Alternatively, the windows analyzed could be small areas of the field of view and could be made to be dynamic, or centered around prior movement, and thus made to track independent movements within the field of view. In this latter case, the computer would need to preview each dynamic window within the field of view and anticipate and store into memory those pixel values which it will need for the subsequent comparison, which may not be coextensive with those pixels used for other comparisons. As shown by the attached software appendix G, fixed pixel sample step sizes are defined at the outset of the subroutine and associated with variables. However, it would be well within the skill of one familiar with writing computer software to implement a subroutine which would adjust the value of the pixel step size variables during the program operation. Implementation of this or other alternate embodiments which make use of dynamic windows would be well within the ordinary level of skill in computer science or electronics.

In addition to overwriting the 2000 samples for each window one-by-one into the computer's RAM, the computer also provides the above-mentioned second monitor output. In the preferred embodiment, the computer is also coupled to a second display monitor 34 that permits observers to observe the operation of the Guest Controlled Orchestra. After the computer has compared the difference between pixel values with 10 hex and F0 hex, the computer also writes a predefined luminance value into the "IVG-128"'s memory associated with the sampled pixel. For example, if the result of the comparison is that the difference is either less than 10 hex or greater F0 hex, a grey value (80 hex) is written into the sampled pixel location. If the pixel change is greater than 10 hex or less than F0 hex, indicating movement, a white value (FF hex) is written into the sampled pixel location.

The "IVG-128" board is configured to independently provide an analog video output that may be used to directly drive a video monitor. Thus, the second video monitor 34 is coupled to this output and requires no intervention by the computer's CPU to display information stored in the "IVG-128"'s memory. As a result of its comparison step, however, the computer changes the sampled pixel value to either white or grey. Thus, on the display monitor, the concentration of white pixels will readily be observed superposed on the black-to-white image as indicating movement, and concentrations of grey pixels observed as indicating regions where movement has not occurred.

Once the computer's CPU has finished a 2000 sample window and has overwritten the last sample into the pixel sample buffer, the CPU will compute the centroid coordinates for that window. The CPU computes a centroid for each window by dividing each of the x index sum and the y index sum for each window by the window's qualifying pixel count. It then stores this centroid coordinate in memory for subsequent use in computing a scalar, described further below. If no qualifying pixels have been found for the window, the CPU will utilize the centroid for the previous frame.

Once both windows have been processed, the CPU satisfies itself that at least four qualifying pixels have been found over both windows. If less than four such pixels have been found, the CPU abandons its centroid and correlation steps and loads a minimum tempo to the MIB, as illustrated in the flow chart of FIG. 4, and then proceeds to process any video frame advance.

After computing the centroid for each window, the CPU will determine a current volume for each channel used of eight possible channels, each corresponding to an orchestral voice.

Each MIDI command either turns an electronic note on or off. The "note on" commands, which are segregated into channels that each correspond to an orchestral voice, also contain digital information as to the volume of the note to be produced. The Guest Controlled Orchestra uses pixel counts to define volume of instruments appearing from each speaker and updates this information as it has finished looking at both windows.

To update the volume corresponding to "note on" commands of each of the eight channels, the CPU looks at each window's pixel count to determine the volume that is to be associated with certain instruments. For example, in the Guest Controlled Orchestra, the left channel is used for bass, and accordingly, the CPU will store a volume level derived from the number of qualifying pixels for the left window into a channel variable corresponding to bass.

MIDI data is formatted to include data representing pitch (128 possible), instrument type (16 possible) and a command that turns the note on or off. Thus, whenever the CPU is called upon to send a "note on" command to the MIB, it simply replaces the bits representing volume with a set of bits derived from the pixel count, depending upon the type of instrument that the particular MIDI data represents. In current use of the Guest Controlled Orchestra, four instruments are used, including flute, bass, marimba, and drum. However, any type or number of instruments can be used. The flute is panned to the right channel and accordingly, the volume bits associated with MIDI flute commands will be derived entirely from the right window changed pixel count and put into a channel variable associated with the flute for use during the one-thirtieth second program cycle. Similarly, the volume of MIDI bass instrument commands will be substituted with a value derived from the changed pixel count in the left window. The changed pixel counts for each of the left and right windows are averaged to yield volume information for marimba and drum note commands. Alternatively, rather than the preferred step of replacing the MIDI command bits associated with note volume, the computer's CPU could be instructed to amplify the command's default volume by an amount dependent upon the changed pixel count.

The system 10 uses two categories of interrupts, which request the CPU to cease performance of all program tasks and perform interim processing. The most significant of these CPU interruptions is utilized to direct filling the MIB's "schedule" of notes to be played. Operation and use of these interrupts may be understood with reference to FIG. 5.

FIG. 5, rather than describing a subroutine or expanded processing step otherwise referred to in FIG. 3, illustrates the flow of the MIB that is distinct and collateral to the main program loop of FIG. 3. When the system is initialized, the MIB has no commands in its buffers, or slots, and the CPU will format and load MIDI commands into eight buffer pairs, defined in RAM by the CPU. Each of these buffer pairs includes a "note on" buffer and a "note off" buffer and corresponds the eight scheduling slots of the MIB. Each of these eight "channels" will correspond to a particular instrument. When the CPU retrieves audio data from RAM and formats it to a MIDI command, the CPU detects the instrument type and places the command into either the "note on" or "note off" buffer corresponding to that instrument. Once the CPU has loaded at least twelve "note on" commands corresponding to each channel into its "note on" buffer (and corresponding "note off" commands into the corresponding "note off" buffer), the CPU sends a command to the MIB directing the MIB to play music.

Operation of the MIB is the initiated by the CPU command, which triggers a generic interrupt from the MIB to tell the CPU that its slots are empty. The CPU feeds a command to the MIB for each channel used, along with a countdown time associated with the command. When the countdown time has expired and the command sent from the MIB to the synthesizer, the MIB generates a channel specific interrupt, which directs the CPU directly to the corresponding "note on" and "note off" channel buffer pair. The CPU looks at each of these buffers to ascertain which buffer holds the command to be issued the soonest. The CPU retrieves this command and formats and sends it to the empty MIB slot. Operation of the Guest Controlled Orchestra currently uses only four of these eight channels, assigned to flute, bass, marimba and drums.

The "slots" described are a set of buffers resident on the MIB that each output a "note on" or "note off" command after a "countdown" time associated with the notes has elapsed. This countdown time is defined in "ticks", with 192 ticks per musical beat. The tempo given to the MIB determines the rate at which the MIB counts "ticks," and hence determines the rate at which notes are played or turned off.

The "MQX-32M" used as the MIB is a standard musical interface board which receives MIDI commands formatted by the computer's CPU, and which puts those commands into one of eight scheduling buffers along with each command's associated countdown time. As stated, when the countdown time associated with the command has elapsed, the MIB removes the command from its associated buffer and feeds the command to the music synthesizer, triggering a channel specific interrupt to the CPU. The MIB thus interrupts the CPU at various intervals seeking a subsequent note command. Once note commands for a particular channel have expired, no new command is sent to the corresponding MIB slot, which remains idle until reactivated by system initialization and a new command, prompted by the generic interrupt, described above.

As the main program loop cycles through every one-thirtieth of a second, the CPU will access its pointer that points to the next digital information in RAM and look at that information stored in the score sequence. Each piece of digital information is stored as a nine byte command, including channel (or orchestral voice), velocity (volume), pitch, note start time and duration. The CPU analyzes channel type and ascertains by looking at the corresponding channel "note on" buffer whether that buffer has its allotment of twelve "note on" commands. If not, the CPU retrieves the nine byte word and increases a pointer to point at the next piece of digital information in the score sequence.

The CPU must define "note on" and "note off" command times from each nine byte word and store respective commands in the channel's associated buffer pair. The constant chosen to define "note on" buffer length defines twelve slots that include an absolute time (four bytes), a default volume, which may be altered as the "note on" command is output from the buffer, and pitch. The "note off" buffer is filled with a corresponding "note off" command, which must be properly inserted into a sequence of "note off" commands since note durations may vary. In other words, since successive note start times will be in sequence, but notes may vary in their duration, associated note end times may be out of sequence. The "note off" buffer for each channel is defined as twenty slots, which each store absolute command time and pitch. The absolute time for the "note off" commands is obtained by summing the absolute time of the corresponding "note on" command with the duration time. Each of these buffers is a circular buffer and has a pointer associated with it by the CPU.

Each command is processed in this manner to fill the CPU's buffers with sufficient notes to last for the next one-thirtieth second cycle.

When the CPU receives a channel specific interrupt from the MIB that tells the CPU to remove a note command from one of its corresponding channel buffer pair, the CPU first compares the next command for each of the "note on" and "note off" buffers corresponding to the channel, or orchestral voice. The command to be executed sooner is removed. If a "note on" command, the volume represented by one of the bytes of the command is overwritten with the current computed volume that has been associated with that channel. Also, the countdown time is computed by the CPU by subtracting the absolute time of the channel's previous command, which is stored by the CPU, from the current absolute time of the "note on" or "note off" command. This countdown time is sent as part of the command to the MIB and the absolute time of the command stored by the CPU for further use.

When the CPU is outputting a command to the MIB and computing the command's countdown time, the CPU compares the countdown time to 240 ticks. This is necessary, since the MIB can only hold a 240 tick countdown time. If the command's time is greater than 240 ticks, the CPU does not increment the note on/off pointer to the chosen command and instead sends an overflow command to the MIB. The CPU retains an absolute time of the overflow command for subsequent comparison with the next command. The overflow command is associated with a 240 tick countdown time which triggers no action by the synthesizer, but which causes another MIB interrupt corresponding to the same channel when the 240 ticks have elapsed.

A mentioned contemplated alternative embodiment also analyzes the guest's actions to select instrument, for example, by analyzing the vertical range of the guests's movement. Thus, the vertical range or location of a guest's action could be made to influence the changing of notes having bass as the default instrument to horn notes, etc. Such a procedure could be implemented, for example, when the CPU removes a command from a channel buffer pair and formats the command for the MIB. This contemplated option has not been implemented in the current system, but it is within the principles of the invention.

With the MIDI command appropriately formatted and stored for output to the MIB from the CPU's buffer, the MIB's interrupt is reset and the CPU returns to its normal program operation.

The MIB operates on supplied tempo information and will employ the most recently provided tempo information to schedule note output. Tempo information is defined as a number of beats per minute, which may vary with the particular music to be played. In the preferred mode of the Guest Controlled Orchestra, the audio data represents a song with 240 beats per minute.

MIDI format specified minimal temporal resolution is 192 ticks per beat. Thus, in the preferred embodiment the MIB will schedule notes with each tick and will adjust its internal clock according to the tempo of the guest's actions in relation to 240 beats per minute. For example, if the guest moves his arms at a fast rate, the tempo will be perceived as being greater than 240 beats per minute, etc. The MIB will read a tempo word fed to it at the end of the main program cycle, every one-thirtieth second, and will perceive changes in the guest's rate of movement, adjusting its own internal clock accordingly. Thus, the MIB counts ticks faster or slower in waiting to output channel commands to the synthesizer, depending upon the guest's rate of motion.

To detect and ascertain this tempo information, the CPU must quantify movement occurring since the previous frame and must correlate this movement to ascertain a pattern of movement. To do this, the CPU looks at the mean location of perceived movement, which will tend to form a path over time. The CPU correlates the position of the current location of movement with the path over time, or history, to determine when it was last at a similar location. Since each location is associated with a frame, or one-thirtieth of a second, the CPU can compute the rate of the guest's movement and a current tempo.

When the CPU has computed the centroids for each of the left and right windows, it computes a scalar, or single value, which is a measure of the magnitude of all movement occurring within both windows. In order to capture all of guest's movement, which may be both vertical and horizontal, the CPU sums all coordinates for all of the centroids within the field of view. Thus, the CPU will add the y index of both the left and right window centroids, as well as the x index. However, since the guest will typically move the left and right arms together vertically, but in opposite directions horizontally, the x index for the centroid for the right window is inverted by taking its 1's complement before summation. In this manner, opposite movements of the left and right arms in the horizontal direction will not cancel each other out, and the resultant scalar will be a measure of both vertical and horizontal movement of both arms.

The CPU stores the scalar for each frame in a two hundred position circular buffer, defined in the computer's RAM. Thus, the computer's CPU will have a long-term memory of the most recent two hundred frames and all older data will be overwritten by the most recent frame's scalar.

When the program is initiated, the CPU's pointer which accesses the MIDI audio data stored in the computer's RAM is initialized to point at the first note of the song. Similarly, the various memory allocations corresponding to buffers to be used with data processing are zeroed. When the computer first detects a guest's presence and subsequent movement, it will load the MIB with a predefined tempo word read from its RAM and corresponding to the ideal rate of play of the music in the computer's RAM. During this time, the CPU analyzes the guest's movements to develop a history and to store this information as scalars in the circular buffer. Once two bars have been played, the CPU is free to change the tempo, as stored in the MIB, and thereby change the rate of play of the song. This "two bar" feature implemented as an inhibit which, after the computer has performed the processing and correlation steps described above and below, will prevent the CPU's update of the sixteen bit tempo word to the MIB.

After the computer's CPU has written the newly processed scalar into the two hundred position circular buffer, it then correlates the most recent thirty scalars over the entire two hundred scalars in memory, as shown in FIG. 5. It does this by utilizing an index to point at the beginning sample of a thirty sample dynamic window over the two hundred scalars, and by performing the following process until one-hundred and seventy iterations have been performed and all two hundred samples correlated. The CPU:

(1) subtracts 30 scalars beginning with the scalar defined by the index from the most recent 30 scalars, respectively;

(2) squares each of the 30 differences obtained and sums all 30 squares together;

(3) stores the sum in a 170 position buffer;

(4) increases the index to point to the next scalar for the next iteration; and

(5) continues with 170 iterations until the index again points to the most recent sample; to save time each iteration subsequent to the first is computed by taking the square of the difference of the oldest scalar within the 30 sample window with the 30th most recent sample, adding that to the most recent sum, and subtracting the square of the difference of the most recent scalar with the newest scalar in the thirty sample window.

This correlation process will thus produce one-hundred and seventy values which are each sums of the squares of the differences between two varying pair groups of thirty samples. Each of these two hundred numbers will vary from a value of zero, i.e., the two thirty sample windows are identical and thus when subtracted produce a value of zero, to a value which may be extremely high, and which is thus provided for by allocating 4 bytes of memory to each of the two hundred memory positions. Thus, the second buffer begins with the value of zero (both thirty sample windows coterminous) and will ascend to a very high number. When a scalar pattern similar to that represented by the most recent frame of data is detected, the sum of the squares of the differences will again be at a minimum. Thus, the positions are analyzed to obtain local minima corresponding to the guest's movements which were similar to that captured by the most recent series of video frames, and tempo extracted.

Since the first of the two hundred summed squares is always zero, the computer's CPU analyzes the increase in the squares by looking for a maximum and a subsequent minimum in two sequential program loops. When it detects a potential maximum, by determining that the next value is less than the preceding value, the CPU applies a bandgap to the maximum to make sure that it is followed by a downward trend differing by at least the number 512. It does this to guard against spurious maxima caused by noise. When it detects a potential minimum, it filters spurious minima by testing subsequent sums to ensure that the potential minimum is followed by an upward trend differing by at least the number 512. The CPU stops searching once it has obtained a second minimum.

Two minima are sought in the preferred embodiment, because it is expected that a guest's arms will be moved both vertically and horizontally. As a result of this movement, the sums of the squares of the differences may experience either one or two minima associated with each cycle of typical motion by the guest. To correctly identify the guest's tempo, the CPU compares the first two minima to ascertain which is smallest, which it presumes to identify the correct tempo. This minimum is compared with a cut-off (selected in the preferred system as the number 2048) to ensure that the detected minimum is small enough to represent a reasonable correlation. If it is not, the correlation result is ignored and the tempo is not updated.

With each correlation, the computer's CPU looks for the first two beats to generate the tempo information. Each squared sum corresponds to a location in the history buffer and is thus associated with time, a particular video frame and a particular point along the scalar's periodic magnitude. The CPU, by performing the correlation process, effectively matches the scalar's change in magnitude at any particular point with a corresponding change of a different cycle, and thereby determines periodicity. The computer's CPU uses this information to compare the beat information derived from periodicity of the guest's movement with the ideal tempo for the particular musical piece. If the guest's tempo deviates from the most recent tempo information that the CPU has, the CPU writes a new 16-bit tempo to the MIB, so that MIDI notes may be appropriately scheduled for output.

The functions described above and embodied in the appended software require mention of several additional points. First, if a guest arrests all movement during the playing of audio data, the software must be capable of freezing the computer's continued correlation of scalars, to avoid skewing the two hundred position circular buffer, or history, described above. In the Guest Controlled Orchestra, the computer's CPU does not perform the correlation steps if the number of changed pixels for a given image frame is less than or equal to four. Thus, the circular buffer will not be updated until the guest once again continues movement. Also, the computer's CPU will arrest the playing of music by the MIB by loading the MIB's tempo register with a minimum tempo, indicating that music is to only be slowly output by the MIB to the MIDI synthesizer.

Also, it is noted that the algorithm utilized in the preferred embodiment for deriving the scalar from the centroids for each frame is susceptible to a minimum tempo if the guest engages in particular movement, i.e., movement of arms that produces a constant scalar as a result of the scalar computation algorithm. It is expected that those skilled in the art can implement an alternative algorithm that avoids this result without departing from the principles of the invention.

In addition, the computer's RAM must store the total number of beats of the chosen musical score. The system 10 preferably uses the song "Under The Sea" from the Disney movie "The Little Mermaid," in a theme park setting. The computer's CPU determines from this information the number of MIDI commands in the audio sequence, and thereby determines when its pointer references the last note in the song. It formats and accordingly loads this final command to the MIB's register, and terminates the main program loop, as indicated in FIG. 3.

The last step of the main program loop, as shown in FIG. 3, is to synchronize the video disk player 30 with the music generated with by the audio system. The CPU determines, from comparison of the tempo to the score's ideal tempo, the rate at which the video disk player 30 is told to advance frames. As video frame advance is directed by the CPU once every thirtieth of a second, the computer will normally direct the video disk player to advance one frame. To the extent that the comparison of the guest's tempo and the ideal tempo are not the same, i.e., their ratio is not an integer, the CPU accumulates any excess to be applied to the next program cycle. Thus, if the guest's tempo is slower than the ideal tempo, the computer will not instruct the video disk player 30 to advance, but will accumulate the guest's beats per minute for addition to the subsequent tempo calculation during the next video frame advance step. In the Guest Controlled Orchestra, the video disk player 30 does not communicate to the computer 20, and hence, the program must instruct the CPU to keep independent tally of the progress of the video frame display sequence.

The disk player used in the system is a "TQ-3032 F" optical disk player, available from Panasonic Industrial Company of Secaucus, New Jersey, and, as mentioned, is coupled to a port of the computer via the RS-232 connector. Each time the video frame advance is called, the computer will either not instruct the video player or it will instruct the video disk player to advance from one-to-five frames. The disk player automatically provides a video rate output of the frame it is instructed to display, and continues displaying that frame, thirty times per second, until instructed to advance by the computer.

The second of the CPU's interrupts is utilized to load data to the RS-232 port for communication to the video disk player. Every one-thirtieth of a second, the CPU will either not command the optical disk player or will format the 1-5 frame advance commands, as discussed. The second interrupt is repeatedly called to load successive individual bytes of this command into an output buffer of the RS-232 port until there are no remaining bytes of the command. After a single byte is loaded, the interrupt is subsequently reset, and the loaded information transmitted bit-by-bit until the output buffer is once again empty. The interrupt will be triggered for loading a remaining byte into the output buffer as long as any bytes in the video command remain.

A further refinement of the system 10 within the scope of the invention would be to implement communication from the video disk player to the computer to indicate frame number. In this manner, the computer 20 could directly read the frame number, instead of keeping independent tally of the frame, as mentioned above. There are commercially available video disk players which have an output port to provide this connection.

Those skilled in the art will observe that numerous changes may be made to the Guest Controlled Orchestra without departing the principles of the current invention.

For example, implementations may be easily devised which correlate values separately for each window, or which simply determine periodicity by analyzing when the scalar's path of movement, or magnitude with respect to time, transgresses a predefined value.

Having thus described several exemplary embodiments of the invention, it will be apparent that various alterations, modifications and improvements will readily occur to those skilled in the art. Such alterations, modifications, and improvements, though not expressly described above, are nonetheless intended and implied to be within the spirit and scope of the invention. Accordingly, the foregoing discussion is intended to illustrative only; the invention is limited and defined only by the following claims and equivalents thereto.

APPENDICES

Appendix A is a "make" file for the programs and routines found in appendices B - G and enables usage directly from the computer's disk operating system.

Appendix B is a software listing in "C" language" of the main program loop.

Appendix C is a machine language listing entitled "COMM.ASM", that handles low level control routines.

Appendix D is a machine language listing entitled "GCO_UTIL.ASM" that includes various utility routines.

Appendix E is a machine language listing containing MIB interface routines.

Appendix F is a machine language listing entitled "SCREEN.ASM" that includes routines for generating 9 X 16 dot matrix characters and controlling display screen functions.

Appendix G is a machine language listing entitled "VIDEO.ASM" that includes image processing routines, routines for computes centroids and scalars, and routines for correlating and analyzing the scalars to provide tempo information.

```
OB\GCO.OBJ: CODE\GCO.C
       tcc -c -f- -C -nob -Icode code\gco.c

OB\MPU401G.OBJ: CODE\MPU401G.ASM
       tasm /mx code\mpu401g, ob\mpu401g;

OB\SCREEN.OBJ: CODE\SCREEN.ASM
       tasm /mx code\screen, ob\screen;

OB\COMM.OBJ: CODE\COMM.ASM
       tasm /mx code\comm, ob\comm;

OB\GCO_UTIL.OBJ: CODE\GCO_UTIL.ASM
       tasm /mx code\gco_util, ob\gco_util;

OB\VIDEO.OBJ: CODE\VIDEO.ASM
       tasm /mx code\video, ob\video;

GCO.EXE: OB\GCO.OBJ OB\MPU401G.OBJ OB\SCREEN.OBJ OB\COMM.OBJ
OB\GCO_UTIL.OBJ OB\VIDEO.OBJ
       tlink @code\m.lnk
```

```
\TURBO\C0S +
OB\GCO +
OB\MPU401G +
OB\SCREEN +
OB\COMM +
OB\GCO_UTIL +
OB\VIDEO
GCO, GCO, \TURBO\CS;
```

```
/*
________________________________________________________________

            WDI Blowpop Controlled Orchestra
          with videodisc visual accompaniment

________________________________________________________________
*/

#include "MUSIC.H"

#define HALT_PERIOD 450
#define MIN_DELTA_PERIOD 10
#define MINIMUM_COUNT 5
#define BANDGAP 4


#define TICK_PER_FRAME 26
            /* ceiling (192 tick/beat  *  240 beat/min  / 1800 frame/min) */
            /* 192*120/900 reduces to 128/5 */
#define TPF_NUMERATOR 128
#define TPF_DENOMINATOR 5
#define TPF_REMAINDER 3
            /* TPF_NUMERATOR - (TPF * TPF_DENOMINATOR) */


int count;

header_rec
  song_header;

extern note
  far *note_0_ptr[];

note
  far *track_mem[8];

meas_rec
  meas[30];

time_rec
  track_time[8][MAX_TIME_CHANGES];

long
  unit_start;

int
   tempo_override,
   velocity_override_0,
   velocity_override_1,
   velocity_override_2,
   velocity_override_3,
   velocity_override_4,
   velocity_override_5,
   velocity_override_6,
        velocity_override_7,

        max_cent,
        min_cent,

        l_x_cent = 0,
        l_y_cent = 0,
        l_count,

        r_x_cent = 0,
        r_y_cent = 0,
        r_count,

        l_volume,
        r_volume,
        m_volume,

        centroid,

        new_period,
        old_period,
        running_period,
        period;

    unsigned int
        avail_mem_size,
        track_mem_size;

    char
        *file_names[] =
            {
            "SONGS\\UT_SEA1.SNG",
            "SONGS\\EKN.SNG",
            "SONGS\\UT_SEA1.SNG",
            "SONGS\\UT_SEA1.SNG",
            },

        file_id[] = "MidiCAD Version 1.00 Song File",

        rising,
        active_tracks,
        message_ready,
        song_is_playing,

        out_string[80],

        velocity_lookup[] =
            {
              0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,  1,  1,  1,  1,
              2,  2,  2,  2,  2,  2,  2,  2,  3,  3,  3,  3,  4,  4,  4,  4,
              5,  5,  6,  6,  7,  7,  8,  8,  9, 10, 11, 12, 13, 14, 15, 16,
             17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32,
             33, 34, 35, 36, 37, 37, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48,
             49, 50, 52, 54, 56, 58, 60, 62, 64, 66, 68, 70, 72, 74, 76, 78,
             80, 82, 84, 86, 88, 90, 92, 94, 96, 98,100,102,104,106,108,110,
            112,114,116,118,120,122,124,126,127,127,127,127,127,127,127,127
            };
        extern void far *dos_malloc();
        extern unsigned _heaplen = 1;

    int
        message_overflow,         /* used for calculating VDP commands */
        message_tally,
        jump_size;

    char
        string_in_use;            /* flag indicating that vdp_command isn't sent yet */
        * vdp_command;            /* command string to be sent to video disk player */

    process_gco_video()
        {
        wait_for_odd_field();

        l_count = get_l_centroid(&l_x_cent, &l_y_cent);
        r_count = get_r_centroid(&r_x_cent, &r_y_cent);

        count = (l_count + r_count) / 2;

        if (count >= MINIMUM_COUNT)
            {
            centroid = (
                (l_x_cent + l_y_cent) * l_count  +
                (r_x_cent + r_y_cent) * r_count
                ) / count / 4;

            compute_correlation(centroid);
            }

        new_period = get_period();

        if(abs(new_period - old_period) < MIN_DELTA_PERIOD)
            running_period = new_period;

        if(new_period != 0)
            old_period = new_period;

        if (count >= MINIMUM_COUNT)
            period = running_period;
        else
            period = HALT_PERIOD;

        display_corr_graphs(centroid, period);
        }

    test_gco()
        {
        while(1)
            {
            init_gco();
            init_graphic_display();

            while(!key_ready())
                process_gco_video();

            if(get_key() == ESC)
            break;

         if(get_key() == ESC)
            break;
         }

      clear_graphics();
      }


play_gco()
   {
   int
      old_tempo = 0,
      i;
   char
      downbeat;        /* hold off flag before giving guest tempo control */

   for(i = 0; i < 8; i++)
      {
      note_0_ptr[i] = track_mem[i];
      if(song_header.track_on_off[i])
         active_tracks |= 1 << i;
      }

   if(active_tracks == 0)
      {
      display_error("No Song Is Loaded, Medfly Maggot !");
      return(0);
      }

   unit_start = 0;

   flush_queues();
   update_queues();

   init_gco();
   init_graphic_display();

   setup_com2();                   /* activate the serial port to the VDP */
   out_str_com2("\2SR1900:\3\13\10");  /* send command to start at frame 1900 */
   while(string_in_use);           /* wait until VPD has command */

   set_tempo(song_header.tempo);
   start_play();

   message_ready = 0;           /* wait two bars before handing over tempo */
   set_clock_to_host(192);      /* one interrupt per beat */
   downbeat = 1;

   while(song_is_playing)
      {
      process_gco_video();

      if(downbeat)
         {
         if (message_ready == 64) /* 4 measures of 4 beats 2 times 2x res */
            {
```

```
            downbeat = 0;
            message_ready = 0;
            set_clock_to_host(TICK_PER_FRAME);
            }
        }
    else if (period != 0)
        {
        if(message_ready > 0)
            {
            /* jump once for each clock tick */
            interrupt_off(); /* interrupt protect this operation */
            jump_size = message_ready;
            message_ready -= jump_size;
            interrupt_on();

            /* convert jump into ticks for precision */
            message_tally += jump_size * TICK_PER_FRAME;

            /* add REMAINDER to tally for each DENOMINATOR counts */
            message_overflow += jump_size;
            if (message_overflow >= TPF_DENOMINATOR)
                {
                message_tally += TPF_REMAINDER;
                message_overflow %= TPF_DENOMINATOR;
                }
            if (!string_in_use)  /* previous message not still pending */
                {

                /* back out correct number of jumps & execute */
                jump_size = message_tally / TICK_PER_FRAME;
                if (jump_size > 5) jump_size = 5;      /* VDP jump limit */
                message_tally -= jump_size * TICK_PER_FRAME;

                switch (jump_size)
                    {
                    case 0:
                        {
                        break;
                        }
                    case 1:
                        {
                        /* command to jump forward 1 frame */
                        out_str_com2("\2JF1:\3\13\10");
                        break;
                        }
                    case 2:
                        {
                        /* command to jump forward 2 frames */
                        out_str_com2("\2JF2:\3\13\10");
                        break;
                        }
                    case 3:
                        {
                        /* command to jump forward 3 frames */
                        out_str_com2("\2JF3:\3\13\10");
                        break;
                        }
                    case 4:
```

```
                    {
                    /* command to jump forward 4 frames */
                    out_str_com2("\2JF4:\3\13\10");
                    break;
                    }
                 case 5:
                    {
                    /* command to jump forward 5 frames */
                    out_str_com2("\2JF5:\3\13\10");
                    break;
                    }
                 }
              }
           }

        if((tempo_override = 3600 / period) > 255)
           tempo_override = 255;

        if (tempo_override != old_tempo)
           {
           set_tempo(tempo_override);
           old_tempo = tempo_override;
           }

        l_volume = l_count * 2;
        r_volume = r_count * 2;
        m_volume =   count * 2;

        l_volume = (l_volume > 127) ? 127 : l_volume;
        r_volume = (r_volume > 127) ? 127 : r_volume;
        m_volume = (m_volume > 127) ? 127 : m_volume;

        velocity_override_0 = velocity_lookup[m_volume];
                                        /* vibe on both speakers */
        velocity_override_1 = velocity_lookup[r_volume];
                                        /* bass, on guest's left speaker */
        velocity_override_2 = velocity_lookup[l_volume];
                                        /* flute, on guest's right speaker *
        velocity_override_3 = velocity_lookup[m_volume];
                                        /* drums on both speakers */
        velocity_override_4 = 0;
        velocity_override_5 = 0;
        velocity_override_6 = 0;
        velocity_override_7 = 0;
        }
     update_queues();

     if(key_ready())
       if(get_key() == ' ')
          break;
     }

  stop_play();
  while(string_in_use);          /* wait to insure that buffer is clear */
  kill_com2();                   /* stop com2 interrupts, message is sent */
  key_wait();
  clear_graphics();
  }
```

```
main()
   {
   int   i;

#ifdef FALSE
   int3();
   setup_com2();              /* activate the serial port to the VDP */

   out_str_com2("\2SR1900:\3\13\10");  /* send command to start at frame 1900 */
   while(string_in_use);      /* wait until VPD has command */

   int3();
   kill_com2();               /* watch the com port closure */

   int3();
   exit(0);
#endif

   init_herc();
   set_graphics();

   init_sparkle_lut();
   stash_int();
   set_mpu_int();
   reset_mpu();
   init_mpu();

   avail_mem_size = get_avail_mem();
   track_mem_size = (avail_mem_size / 8) - 1;

   for(i = 0; i < 8; i++)
      track_mem[i] = dos_malloc(track_mem_size);

   new_screen:
   display_centered("W  D  I", 0);
   display_centered("G U E S T", 2);
   display_centered("C O N T R O L L E D", 3);
   display_centered("O R C H E S T R A", 4);

   display_centered("Available Selections", 8);
   display_centered("1 - UNDER THE SEA", 10);
   display_centered("2 - NACHT MUSIC", 12);
   display_centered("3 - UNDER THE SEA", 14);
   display_centered("4 - UNDER THE SEA", 16);

   display_centered("Whack The Spacebar To Start, Dude", 19);

   display_centered("A - Align Camera      T - Test Without Music", 23);

   while(1)
      {
      active_tracks = 0;

      switch(get_key())
         {
         case ESC:
```

```
                    fix_int();
                    set_text();
                    exit(0);

                 case 'A':
                 case 'a':
                    init_norm_lut();
                    align_camera();
                    init_sparkle_lut();
                 break;

                 case 'T':
                 case 't':
                    test_gco();
                 goto new_screen;

                 case '1':
                    load_file(0);
                 break;

                 case '2':
                    load_file(1);
                 break;

                 case '3':
                    load_file(2);
                 break;

                 case '4':
                    load_file(3);
                 break;

                 case ' ':
                    play_gco();
                 goto new_screen;
                 }
              }
           }


   load_file(index)
   int   index;
      {
      int
         i,
         meas_size,
         time_size,
         byte_count,
         header_size,
         filehandle;

      if((filehandle = _open(file_names[index], 0)) == -1)
         {
         display_error("Unable To Open File");
         return(0);
         }

      if(_read(filehandle, out_string, sizeof(file_id)) != sizeof(file_id))
```

```
            goto read_error;

         if(strcmp(out_string, file_id))
            {
            display_error("Invalid File");
            _close(filehandle);
            return(0);
            }

         if(_read(filehandle, &song_header, sizeof(song_header)) != sizeof(song_header)
            goto read_error;

         meas_size = song_header.num_meas_names * sizeof(meas_rec);

         if(_read(filehandle, &meas[0], meas_size) != meas_size)
            goto read_error;

         for(i = 0; i < 8; i++)
            {
            time_size = song_header.num_times[i] * sizeof(time_rec);
            if(_read(filehandle, &track_time[i][0], time_size) != time_size)
               goto read_error;

            byte_count = song_header.track_size[i] * sizeof(note);
            if(far_read(filehandle, track_mem[i], byte_count) != byte_count)
               goto read_error;
            }

        _close(filehandle);
        return(0);

        read_error:
        display_error("Unable To Read File");
        _close(filehandle);
        return(0);
        }

   init_gco()
      {
      int i;

      wait_for_odd_field();
      init_history_buffers();

      velocity_override_0 =
      velocity_override_1 =
      velocity_override_2 =
      velocity_override_3 =
      velocity_override_4 =
      velocity_override_5 =
      velocity_override_6 =
      velocity_override_7 = 32;

      l_volume =
      r_volume =
      m_volume = 64;

      rising = 1;
```

```
    min_cent = 30000;
    max_cent = 0;

    period = new_period = old_period = running_period = 0;
    }

init_graphic_display()
    {
    clear_graphics();
    display_text("Filtered X Left Centroid", 0, 6, 1);
    display_text("Running Pixel Correlation", 0, 13, 1);
    init_corr_graphs();
    }

display_error(string)
char string[];
    {
    display_blanks(80, 0, 24);
    display_text(string, 0, 24);
    key_wait();
    display_blanks(80, 0, 24);
    return(0);
    }

display_message(string)
char string[];
    {
    display_blanks(80, 0, 24);
    display_text(string, 0, 24);
    return(0);
    }

display_centered(string, line)
char string[];
int line;
    {
    display_blanks(80, 0, line);
    display_text(string, (80 - strlen(string)) / 2, line);
    }
```

```
                    Serial Communication Module   COMM.ASM

                          Includes routines for:
                       Low level control for COM1 or COM2

           ALL ROUTINES CONTROL THE UARTS DIRECTLY AT THE I/O PORT LEVEL

*
COM1_DATA          equ 3F8h                   ;COM1 transceiver data register
COM1_IER           equ COM1_DATA+1            ;COM1 interrupt enable register
COM1_IIR           equ COM1_DATA+2            ;COM1 interrupt indentification reg
COM1_LCR           equ COM1_DATA+3            ;COM1 line condition register
COM1_MCR           equ COM1_DATA+4            ;COM1 modem control register
COM1_LSR           equ COM1_DATA+5            ;COM1 line status register
COM1_MSR           equ COM1_DATA+6            ;COM1 modem status register

COM2_DATA          equ 2F8h                   ;COM2 transciever data register
COM2_IER           equ COM2_DATA+1            ;COM2 interrupt enable register
COM2_IIR           equ COM2_DATA+2            ;COM2 interrupt indentification reg
COM2_LCR           equ COM2_DATA+3            ;COM2 line condition register
COM2_MCR           equ COM2_DATA+4            ;COM2 modem control register
COM2_LSR           equ COM2_DATA+5            ;COM2 line status register
COM2_MSR           equ COM2_DATA+6            ;COM2 modem status register

;for MCR (modem control reg)
OUT2_BIT           equ 00001000B
RTS_BIT            equ 00000010B
DTR_BIT            equ 00000001B

;for MSR (modem status reg)
DCD_MASK           equ 10000000B
RI_MASK            equ 01000000B
DSR_MASK           equ 00100000B
CTS_MASK           equ 00010000B
DCD_DELTA_MASK     equ 00001000B
RI_DELTA_MASK      equ 00000100B          ;H -> L only
DSR_DELTA_MASK     equ 00000010B
CTS_DELTA_MASK     equ 00000001B

;for LSR (line status reg)
TXDE_MASK          equ 00100000B          ;transmit data register empty
FE_MASK            equ 00001000B          ;frame error
PE_MASK            equ 00000100B          ;parity error
OE_MASK            equ 00000010B          ;overrun error (data lost)
RXDA_MASK          equ 00000001B          ;receive data available

;for IER (interrupt enable register)
MODEM_INT_EN       equ 00001000B          ;modem interrupt enable
RXIE_INT_EN        equ 00000100B          ;rx interrupt enable
TXDE_INT_EN        equ 00000010B          ;tx data empty interrupt enable
```

```
RXDA_INT_EN     equ 00000001B             ;rx data available interrupt enable

;for IIR (interrupt identification register)
IPN             equ 00000001B             ;active low, interrupt pending flag
RX_ERROR_CON    equ 00000110B             ;rx error condition
RX_CHAR_AVAIL   equ 00000100B             ;rx character available
TXD_REG_EMPTY   equ 00000010B             ;tx data register empty
MODEM_INT       equ 00000000B             ;modem lines interrupt

;for LCR
DLAB_C          equ 01111111B             ;divisor latch access bit (mask)

;for INT14
BAUD110         equ 00000000B
BAUD150         equ 00100000B
BAUD300         equ 01000000B
BAUD600         equ 01100000B
BAUD1200        equ 10000000B
BAUD2400        equ 10100000B
BAUD4800        equ 11000000B
BAUD9600        equ 11100000B

PARITYNO        equ 00000000B
PARITYODD       equ 00001000B
PARITYEVEN      equ 00011000B

STOP1           equ 00000000B
STOP2           equ 00000100B

DATA5           equ 00000000B
DATA6           equ 00000001B
DATA7           equ 00000010B
DATA8           equ 00000011B

;return values
NO_ERROR        equ 0
UART_ERROR      equ 1

;8259 values
EOI             equ 20h
PORT8259        equ 20h
INT_MASK        equ 21h                   ;interrupt mask register (active low)

                .model small

                .data?

old_int_0B      dd 0
out_string_com2 dd 0                      ;seg & off to string sent by routine

                extrn _string_in_use:BYTE

                .code

;*************************************************************************
;*************************************************************************
;
;                    void interrupt_on() & interrupt_off()
```

```
;
;                        Enables/Disables interrupts
;
;*****************************************************************************
;*****************************************************************************

                 public _interrupt_off
_interrupt_off   proc near
                 cli
_interrupt_off   endp

                 public _interrupt_on
_interrupt_on    proc near
                 sti
_interrupt_on    endp

;*****************************************************************************
;*****************************************************************************
;
;                          void setup_com2()
;
;                 Sets Communication Parameters For COM2
;                        Clears receive buffer
;                       Installs interrupt vector
;        Enables interrupt for character transmission buffer empty
;
;*****************************************************************************
;*****************************************************************************

                 public _setup_com2
_setup_com2      proc near

                 mov dx,1                     ;COM2 (for COM1 is 0)
                 mov ah,0
                 mov al,BAUD9600+PARITYNO+STOP1+DATA8
                 int 14H

                 mov dx,COM2_DATA             ;Reading the receive register clears it
                 in al,dx

                 mov al,0Bh                   ;save old IRQ3 (INT 0Bh)
                 mov ah,35h                   ;code for get vector
                 int 21h
                 mov word ptr old_int_0B,bx ;stash old vector away
                 mov word ptr old_int_0B[2],es

                 push ds                      ;set COM2 interrupt vector

                 mov dx,seg out_char_com2
                 mov ds,dx
                 mov dx,offset out_char_com2

                 mov al,0Bh
                 mov ah,25h
                 int 21h

                 pop ds
```

```
                cli

                mov dx,COM2_IER         ;allow transmit data empty interrupts
                mov al,TXDE_INT_EN
                out dx,al

                mov dx,COM2_IIR         ;clear first xmit data empty
                in al,dx

                mov dx,COM2_MCR         ;enable card's interrupts
                mov al,OUT2_BIT
                out dx,al               ; ---secret knowledge---

                in al,INT_MASK          ;Get mask
                and al,11110111B        ;enable IRQ 3
                out INT_MASK,al         ;put it back

                mov al,EOI              ;clock the 8259
                out PORT8259,al

                sti

                ret

_setup_com2     endp

;****************************************************************************
;****************************************************************************
;
;                          void kill_com2()
;
;               Halts Communication Parameters For COM2
;                    Disables COM2 interrupts
;                    Removes interrupt vector
;
;****************************************************************************
;****************************************************************************

                public _kill_com2
_kill_com2      proc near

                mov al,0
                mov dx,COM2_MCR
                out dx,al               ;clear the OUT2, RTS & DTR lines

                cli

                in al,INT_MASK          ;Get mask
                or al,00001000B         ;disable IRQ 3
                out INT_MASK,al         ;put it back

                mov al,EOI              ;clock 8259
                out PORT8259,al

                sti

                push ds                 ;restore old vector
                lds dx,old_int_0B
                mov al,0Bh
                mov ah,25h
                int 21h

                pop ds

                ret

_kill_com2      endp

;***************************************************************************
;***************************************************************************
;
;               void _out_str_com2(string: char *)
;                                          [bp+4]
;               sends a \0 terminated string out via com2
;
;***************************************************************************
;***************************************************************************

                public _out_str_com2
_out_str_com2   proc near

                push bp
                mov bp,sp

                mov bx,[bp+4]               ;get pointer to first character
                mov al,[bx]                 ;dereference once
                cmp al,0                    ;test first character
                jz out_str_com2_9           ;bif null string

                mov _string_in_use,1        ;set "string in use" flag

                inc bx                      ;already used first character, now sec.
                mov WORD PTR out_string_com2,bx   ;save updated pointer
                mov WORD PTR out_string_com2+2,ds

                mov dx,COM2_DATA
                out dx,al                   ;send character, further ones with INT

out_str_com2_9:
                pop bp

                ret

_out_str_com2   endp

;***************************************************************************
;***************************************************************************
;

;               out_char_com2
;               sends next character of string when TXDE interrupt occurs
;               NOTE: interrupt vector routine!!!
;               assumes only IRQ3 (INT0B) interrupts come from need for TX data
```

```
;
;*****************************************************************************
;*****************************************************************************

out_char_com2  proc near

               push ds
               push dx
               push bx
               push ax

               mov dx,COM2_IIR             ;identify interrupt source
               in al,dx
               cmp al,TXD_REG_EMPTY        ;test for intended interrupt (TxDE)
               jne out_char_com2_9         ;bif not the correct kind

               mov al,EOI                  ;clear the 8259
               out PORT8259,al

               mov bx,seg out_string_com2     ;load character pointer address
               mov ds,bx
               push ds                     ;additional save for later...

               lds bx,out_string_com2      ;get character pointer

               mov al,[bx]                 ;get character

               cmp al,0
               pop ds
               je out_char_com2_8          ;bif last character gone

               inc word ptr out_string_com2 ;point to next character

               mov dx,COM2_DATA
               out dx,al                   ;send character, further ones with INT

               jmp out_char_com2_9         ;jump to end

out_char_com2_8:
               mov bx,seg _string_in_use
               mov ds,bx
               mov _string_in_use,0        ;clear "string in use" flag

out_char_com2_9:
               pop ax
               pop bx
               pop dx
               pop ds

               iret

out_char_com2  endp

               end
```

```
;
;                    Various Assembly Utility Routines UTIL.ASM
;
;_________________________________________________________________________
;
;                        WDI Guest Controlled Orchestra
;
;
;
;_________________________________________________________________________
;

                .model small
                .code

;*************************************************************************

;*************************************************************************

                public _key_ready
_key_ready      proc near

                mov ah,1
                int 16H
                jz no_key

                mov ax,1
                ret
no_key:
                mov ax,0
                ret

_key_ready      endp

;*************************************************************************

;*************************************************************************

                public _get_key
_get_key        proc near

                mov ah,0
                int 16H
                cmp al,0
                jz is_ext

                mov ah,0
is_ext:
                ret

_get_key        endp

;*************************************************************************

;*************************************************************************

                public _key_wait
_key_wait       proc near

                mov ah,0
                int 16H
                ret

_key_wait       endp

;***************************************************************************
;*
;*
;*              void far *dos_malloc(unsigned int)
;*              pointer = dos_malloc(paragraph_count);
;***************************************************************************

                public _get_avail_mem
_get_avail_mem  proc near

                mov bx,0FFFFH                   ;request excessive memory
                mov ah,48H                      ;DOS allocate function
                int 21H

                mov ax,bx                       ;AX has largest available block
                ret

_get_avail_mem endp

;***************************************************************************
;*
;*
;*              void far *dos_malloc(unsigned int)
;*              pointer = dos_malloc(paragraph_count);
;***************************************************************************

                public _dos_malloc
_dos_malloc     proc near

                push bp
                mov bp,sp

                mov bx,[bp+4]                   ;requested memory in paragraphs
                mov ah,48H                      ;DOS allocate function
                int 21H
                jnc alloc_ok

                mov dx,0                        ;Alloc failed, return NULL pointer
                mov ax,0

                jmp alloc_end
alloc_ok:
                mov dx,ax                       ;Alloc OK, return far pointer to mem
                mov ax,0

alloc_end:
                pop bp
                ret

_dos_malloc     endp
```

```
;*******************************************************************************
;*******************************************************************************

                public _far_read
_far_read       proc near

                push bp
                mov bp,sp

                push ds

                mov bx,[bp+4]
                lds dx,[bp+6]
                mov cx,[bp+10]
                mov ah,3FH
                int 21H

                jnc read_f_ok
                mov ax,-1
read_f_ok:
                pop ds
                pop bp
                ret

_far_read       endp

                end
```

```
;
;                    MPU401 Interface Routines MPU401G.ASM
;_____________________________________________________________________________
;
;                      WDI Guest Controlled Orchestra
;
;
;_____________________________________________________________________________
;

QUEUE_SIZE          equ 12                   ;max notes in each queue
MAX_OFF             equ 20                   ;max notes on per track

EOI                 equ 20H                  ;code for end of interrupt
INT_CMD             equ 20H                  ;interrupt controller command register
INT_MASK            equ 21H                  ;and mask register

DATA_PORT           equ 330H                 ;The MPU-401 IO Ports
STAT_PORT           equ 331H

DRR                 equ 40H                  ;The MPU-401 Handshake Lines
DSR                 equ 80H

NOTE_ON             equ 90H
MAX_TIME            equ 240

START_PLAY          equ 0AH
STOP_PLAY           equ 05H

NO_REAL_TIME_OUT          equ  32H
SEND_MEASURE_END_OFF      equ  8CH
CLOCK_TO_HOST_OFF         equ  94H
CLOCK_TO_HOST_ON          equ  95H
CLEAR_PLAY_COUNTERS       equ 0B8H
SET_TIMEBASE              equ 0C8H
SET_TEMPO                 equ 0E0H
SET_CLOCK_TO_HOST         equ 0E7H
ACTIVATE_TRACKS           equ 0ECH
TIMING_OVERFLOW           equ 0F8H
DATA_END                  equ 0FCH
ACK                       equ 0FEH
RESET                     equ 0FFH

                .model small

;***************************************************************************
;*                      Structure Of Raw Note Data                         *
;***************************************************************************

note            struc

start_lo        dw 0                  ; Absolute, In MPU Clocks (Low Word)
start_hi        dw 0                  ; Absolute, In MPU Clocks (High Word)
duration        dw 0                  ; In MPU Clocks
pitch           db 0                  ; Midi Pitch     0 to 127
velocity        db 0                  ; Midi Velocity 0 to 127
```

```
channel          db 0                         ; Midi Channel  0 to 15

note             ends

;*****************************************************************************
;*                  For Pointer To Note On Waiting Data                      *
;*****************************************************************************

on_ptr           struc

time_to_on_lo    dw 0
time_to_on_hi    dw 0
on_channel       db 0
on_note          db 0
on_velocity      db 0

on_ptr           ends

;*****************************************************************************
;*                  For Pointer To Note Off Waiting Data                     *
;*****************************************************************************

off_ptr          struc

time_to_off_lo   dw 0
time_to_off_hi   dw 0
off_channel      db 0
off_note         db 0

off_ptr          ends

;*****************************************************************************
;*                       For Pointer To Track Queues                         *
;*****************************************************************************

queue_ptr        struc

timing_byte      db 0
midi_command     db 0
midi_note        db 0
midi_velocity    db 0

queue_ptr        ends

;*****************************************************************************
;*                                Macros
;*****************************************************************************

wait_for_dsr     macro
                 local wait_loop
                 mov dx,STAT_PORT
wait_loop:
                 in al,dx
                 and al,DSR
                 cmp al,DSR
                 je wait_loop
                 endm
```

```
;***************************************************************************
;*                        Initialized Variables                            *
;***************************************************************************

                .data

                extrn _active_tracks:byte
                extrn _song_is_playing:byte
                extrn _unit_start:dword

active_queues   db 0
cur_track       db 0
time_to_event   dw 0
target          dw 0
queue_updated   dw 0
old_mask        db 0

extrn           _message_ready:byte

extrn           _velocity_override_0:word
extrn           _velocity_override_1:word
extrn           _velocity_override_2:word
extrn           _velocity_override_3:word
extrn           _velocity_override_4:word
extrn           _velocity_override_5:word
extrn           _velocity_override_6:word
extrn           _velocity_override_7:word

;***************************************************************************
;*                Jump Table For Executing MPU Commands                    *
;***************************************************************************

mpu_messages    dw offset track0_data_request
                dw offset track1_data_request
                dw offset track2_data_request
                dw offset track3_data_request
                dw offset track4_data_request
                dw offset track5_data_request
                dw offset track6_data_request
                dw offset track7_data_request
                dw offset timing_data_overflow
                dw offset conductor_data_request
                dw offset undefined1
                dw offset undefined2
                dw offset all_end
                dw offset clock_to_host
                dw offset is_ack
                dw offset system_message

                .data?

running_status db ?

;***************************************************************************
;*                          Track Timers                                   *
```

```
;***************************************************************************
timer           dw ?                    ;Pointer to timers

timer_0         dd ?
timer_1         dd ?
timer_2         dd ?
timer_3         dd ?
timer_4         dd ?
timer_5         dd ?
timer_6         dd ?
timer_7         dd ?

;***************************************************************************
;*                    Pointers To The Raw Note Data                        *
;***************************************************************************

                public _note_0_ptr
_note_0_ptr     dd ?

note_1_ptr      dd ?
note_2_ptr      dd ?
note_3_ptr      dd ?
note_4_ptr      dd ?
note_5_ptr      dd ?
note_6_ptr      dd ?
note_7_ptr      dd ?

;***************************************************************************
;*                 Pointers To Heads Of Track Queues                       *
;***************************************************************************

track_0_head    dw ?
track_1_head    dw ?
track_2_head    dw ?
track_3_head    dw ?
track_4_head    dw ?
track_5_head    dw ?
track_6_head    dw ?
track_7_head    dw ?

;***************************************************************************
;*                 Pointers To Tails Of Track Queues                       *
;***************************************************************************

track_0_tail    dw ?
track_1_tail    dw ?
track_2_tail    dw ?
track_3_tail    dw ?
track_4_tail    dw ?
track_5_tail    dw ?
track_6_tail    dw ?
track_7_tail    dw ?

;***************************************************************************
;*                    The Track Queues Themselves                          *
;***************************************************************************
```

```
track_0_base   db size queue_ptr * QUEUE_SIZE dup (?)
track_1_base   db size queue_ptr * QUEUE_SIZE dup (?)
track_2_base   db size queue_ptr * QUEUE_SIZE dup (?)
track_3_base   db size queue_ptr * QUEUE_SIZE dup (?)
track_4_base   db size queue_ptr * QUEUE_SIZE dup (?)
track_5_base   db size queue_ptr * QUEUE_SIZE dup (?)
track_6_base   db size queue_ptr * QUEUE_SIZE dup (?)
track_7_base   db size queue_ptr * QUEUE_SIZE dup (?)
last_base      db ?

;*****************************************************************************
;*                  Note On Commands Waiting To Go On Queue                  *
;*                                                                           *
;*                                  Format                                   *
;*   on_note:byte, time_till_on:word, on_velocity:byte, on_channel:byte      *
;*****************************************************************************

track_0_on     db size on_ptr dup (?)
track_1_on     db size on_ptr dup (?)
track_2_on     db size on_ptr dup (?)
track_3_on     db size on_ptr dup (?)
track_4_on     db size on_ptr dup (?)
track_5_on     db size on_ptr dup (?)
track_6_on     db size on_ptr dup (?)
track_7_on     db size on_ptr dup (?)

;*****************************************************************************
;*                  Note Off Commands Waiting To Go On Queue                 *
;*                Up To MAX_OFF Notes Can Be Waiting Per Track               *
;*                                                                           *
;*     Format   off_note:byte, time_till_off:word, off_channel:byte          *
;*****************************************************************************

track_0_off    db MAX_OFF * size off_ptr dup (?)
track_1_off    db MAX_OFF * size off_ptr dup (?)
track_2_off    db MAX_OFF * size off_ptr dup (?)
track_3_off    db MAX_OFF * size off_ptr dup (?)
track_4_off    db MAX_OFF * size off_ptr dup (?)
track_5_off    db MAX_OFF * size off_ptr dup (?)
track_6_off    db MAX_OFF * size off_ptr dup (?)
track_7_off    db MAX_OFF * size off_ptr dup (?)

               .code

old_int_0f     dd 0

;*****************************************************************************
;*                   Compute A Note And Put It In Queue                      *
;*                                                                           *
;*                 Assumes: SI Points to 'note on' wait list                 *
;*                          DI Points to 'note off' wait list                *
;*                          BX Points to raw note data                       *
;*                          ES Points to raw note seg                        *
;*****************************************************************************

compute_note   proc near
```

```
                push ax                     ;save queue pointer

                mov queue_updated,1
                cmp [si].on_channel, 0FFH ;illegal channel means no note on wait:
                jz no_note_on_waiting
                jmp compare_times

;************************** Check For End Of Notes **************************

no_note_on_waiting:
                cmp byte ptr es:[bx].channel, 0FFH ;illegal channel means no note
left
                jnz get_new_note            ;if new notes are waiting, do em

                cmp [di].off_channel, 0FFH ;illegal channel means no note off wa:
                jz end_compute
                jmp do_off                  ;if note off is waiting, do it

end_compute:
                jmp is_end                  ;if no new notes and no offs waiting

;************* Get New Note And Put It Onto Note On Waiting List **************
;******************** Put Its Stop Time On Note Off List **********************

get_new_note:
                mov ax,es:[bx].start_lo
                mov dx,es:[bx].start_hi

                push di
                mov di, timer
                sub ax,[di]                 ;Compute time to on in Long Int
                jnc nc1
                dec dx
nc1:
                sub dx,[di+2]
                pop di

                mov [si].time_to_on_lo,ax
                mov [si].time_to_on_hi,dx
                add ax,es:[bx].duration     ;duration + time to on = time to off
                jnc nc2
                inc dx
nc2:

                mov cl,es:[bx].channel
                mov [si].on_channel,cl

                mov cl,es:[bx].pitch
                mov [si].on_note,cl

                mov cl,es:[bx].velocity
                and cl,07FH                 ;mask out marking
                mov [si].on_velocity,cl

;******************** Restructure The 'Off Wait' List ***************

                push di
search_loop:
```

```
                    cmp dx,[di].time_to_off_hi
                    ja search_more

                    cmp ax,[di].time_to_off_lo
                    jb insert_it
search_more:
                    add di, size off_ptr
                    jmp search_loop
insert_it:
                    mov target, di

                    cmp [di].off_channel, 0FFH ;illegal channel means no note off wai
                    jz insert_data

;************************ Find The End Of The List ************************

end_loop:
                    cmp [di].off_channel, 0FFH ;illegal channel means no note off wai
                    jz space_loop

                    add di, size off_ptr
                    jmp end_loop

;************************ Make Space For The New Data ************************

space_loop:
                    mov cl,[di-size off_ptr].off_note
                    mov [di].off_note,cl

                    mov cx,[di-size off_ptr].time_to_off_lo
                    mov [di].time_to_off_lo,cx

                    mov cx,[di-size off_ptr].time_to_off_hi
                    mov [di].time_to_off_hi,cx

                    mov cl,[di-size off_ptr].off_channel
                    mov [di].off_channel,cl

                    sub di, size off_ptr
                    cmp di,target
                    jnz space_loop

insert_data:

;************************ Insert The New Data ************************

                    mov cl,es:[bx].channel
                    mov [di].off_channel,cl

                    mov cl,es:[bx].pitch
                    mov [di].off_note,cl

                    mov [di].time_to_off_lo,ax
                    mov [di].time_to_off_hi,dx

                    pop di
                    add bx,size note           ;point to next raw note
```

```
compare_times:
                cmp [di].off_channel, OFFH ;illegal channel means no note off wa
                jz do_on                   ;if no off is waiting, process the on

                mov ax,[si].time_to_on_hi     ;which one happens first?
                cmp ax,[di].time_to_off_hi
                jb do_on

                mov ax,[si].time_to_on_lo
                cmp ax,[di].time_to_off_lo
                jb do_on

;******************** Put A 'Note Off' Onto The Queue ****************

do_off:
                mov ax,bx                  ;save raw note pointer
                pop bx                     ;get back queue pointer
                push ax                    ;save that note pointer

                cmp [di].time_to_off_hi,0
                jnz is_overflow

                mov ax,[di].time_to_off_lo
                cmp ax, MAX_TIME
                jae is_overflow

                mov time_to_event, ax
                mov [bx].timing_byte,al    ;put time into queue

                mov al,[di].off_note
                mov [bx].midi_note,al      ;and note

                mov al,[di].off_channel
                or al,NOTE_ON
                mov [bx].midi_command,al   ;and command

                mov [bx].midi_velocity,0   ;note_off = note_on with v = 0

;******************** Restructure The 'Off Wait' List ***************

                push di

fixup_loop:
                mov al,[di+size off_ptr].off_note
                mov [di].off_note,al

                mov ax,[di+size off_ptr].time_to_off_lo
                mov [di].time_to_off_lo,ax

                mov ax,[di+size off_ptr].time_to_off_hi
                mov [di].time_to_off_hi,ax

                mov al,[di+size off_ptr].off_channel
                mov [di].off_channel,al

                cmp [di].off_channel, OFFH ;illegal channel means no note off wait
                jz fixup_done
                add di, size off_ptr
                jmp fixup_loop

fixup_done:
                pop di
                jmp process_events

;******************** Put A 'Note On' Onto The Queue *****************

do_on:
                mov ax,bx                       ;save raw note pointer
                pop bx                          ;get back queue pointer
                push ax                         ;save that note pointer

                cmp [si].time_to_on_hi,0
                jnz is_overflow

                mov ax,[si].time_to_on_lo
                cmp ax,MAX_TIME
                jae is_overflow

                mov time_to_event, ax
                mov [bx].timing_byte,al         ;put time into queue

                mov al,[si].on_note
                mov [bx].midi_note,al           ;and note

                mov al,[si].on_channel
                or al,NOTE_ON
                mov [bx].midi_command,al        ;and command

                mov al,[si].on_velocity
                mov [bx].midi_velocity,al       ;and velocity

                mov [si].on_channel, 0FFH ;illegal channel means no note on waitir

                jmp process_events

is_overflow:
                mov [bx].timing_byte,TIMING_OVERFLOW
                mov time_to_event, MAX_TIME
                jmp process_events

is_end:
                mov ax,bx                       ;save raw note pointer
                pop bx                          ;get back queue pointer
                push ax                         ;save that note pointer

                mov [bx].timing_byte,0
                mov [bx].midi_command,DATA_END

                mov al,cur_track
                xor active_queues,al            ;turn the queue off
                jmp end_compute_note

process_events:
```

```
;************************** Increment Timer ****************************

                mov bx, timer
                mov ax,time_to_event      ;doubleword inc timer
                add [bx],ax
                jnc no_carry
                inc word ptr[bx+2]
no_carry:

;******************** Decrement All Waiting Events **********************

                cmp [si].on_channel, 0FFH ;illegal channel means no note on waiti
                jz dec_note_offs

                sub [si].time_to_on_lo,ax
                jnc dec_note_offs
                dec [si].time_to_on_hi

dec_note_offs:
                cmp [di].off_channel, 0FFH ;illegal channel means no note off wai
                jz end_compute_note

                sub [di].time_to_off_lo,ax
                jnc nc3
                dec [di].time_to_off_hi
nc3:

                add di, size off_ptr
                jmp dec_note_offs

end_compute_note:
                pop ax                     ;return raw note pointer in ax
                ret

compute_note    endp

;*************************************************************************
;*                      Flush The Track Queues                           *
;*************************************************************************

                public _flush_queues
_flush_queues   proc near

                push di

                mov al,_active_tracks
                mov active_queues,al

                mov ax, word ptr _unit_start
                mov dx, word ptr _unit_start[2]

                mov word ptr timer_0, ax
                mov word ptr timer_0[2], dx

                mov word ptr timer_1, ax
                mov word ptr timer_1[2], dx
                mov word ptr timer_2, ax
                mov word ptr timer_2[2], dx

                mov word ptr timer_3, ax
                mov word ptr timer_3[2], dx

                mov word ptr timer_4, ax
                mov word ptr timer_4[2], dx

                mov word ptr timer_5, ax
                mov word ptr timer_5[2], dx

                mov word ptr timer_6, ax
                mov word ptr timer_6[2], dx

                mov word ptr timer_7, ax
                mov word ptr timer_7[2], dx

                mov track_0_head,offset dgroup:track_0_base
                mov track_1_head,offset dgroup:track_1_base
                mov track_2_head,offset dgroup:track_2_base
                mov track_3_head,offset dgroup:track_3_base
                mov track_4_head,offset dgroup:track_4_base
                mov track_5_head,offset dgroup:track_5_base
                mov track_6_head,offset dgroup:track_6_base
                mov track_7_head,offset dgroup:track_7_base

                mov track_0_tail,offset dgroup:track_0_base
                mov track_1_tail,offset dgroup:track_1_base
                mov track_2_tail,offset dgroup:track_2_base
                mov track_3_tail,offset dgroup:track_3_base
                mov track_4_tail,offset dgroup:track_4_base
                mov track_5_tail,offset dgroup:track_5_base
                mov track_6_tail,offset dgroup:track_6_base
                mov track_7_tail,offset dgroup:track_7_base

                mov ax,dgroup
                mov es,ax
                lea di,track_0_on

                mov al,0FFH
                mov cx, 8 * size on_ptr + 8 * MAX_OFF * size off_ptr

                rep stosb

                pop di

_flush_queues   endp

;**************************************************************************
;*              Check The Track Queues And Add Notes If Necessary          *
;**************************************************************************

                public _update_queues
_update_queues  proc near

                push si
                push di
queue_loop:
                mov queue_updated,0

;******************** Update Track 0 Queue ***************************

                test active_queues,1
                jz check_track_1

                mov cur_track,1
                mov ax,track_0_tail
                mov bx,size queue_ptr
                add bx,ax
                cmp bx,offset dgroup:track_1_base ;at end of array
                jnz not_end_0

                lea bx,track_0_base ;if at end, point to start
not_end_0:
                cmp bx,track_0_head        ;if tail + size = head, queue is full
                jz check_track_1

                lea si,track_0_on         ;point to on and off wait lists
                lea di,track_0_off
                les bx,_note_0_ptr          ;and note data

                lea dx, timer_0
                mov timer, dx

                call compute_note          ;with ax = queue pointer
                mov word ptr _note_0_ptr,ax  ;update raw note pointer

                mov ax, track_0_tail
                add ax, size queue_ptr

                cmp ax,offset dgroup:track_1_base ;at end of array
                jnz nott_end_0

                lea ax,track_0_base ;if at end, point to start
nott_end_0:
                mov track_0_tail, ax

;******************** Update Track 1 Queue ***************************

check_track_1:

                test active_queues,2
                jz check_track_2

                mov cur_track,2
                mov ax,track_1_tail
                mov bx,size queue_ptr
                add bx,ax
                cmp bx,offset dgroup:track_2_base ;at end of array
                jnz not_end_1
                lea bx,track_1_base ;if at end, point to start
not_end_1:
                cmp bx,track_1_head       ;if tail + size = head, queue is full
                jz check_track_2

                lea si,track_1_on  ;point to on and off wait lists
                lea di,track_1_off
                les bx,note_1_ptr         ;and note data

                lea dx, timer_1
                mov timer, dx

                call compute_note         ;with ax = queue pointer
                mov word ptr note_1_ptr,ax  ;update raw note pointer

                mov ax, track_1_tail
                add ax, size queue_ptr

                cmp ax,offset dgroup:track_2_base ;at end of array
                jnz nott_end_1

                lea ax,track_1_base ;if at end, point to start
nott_end_1:
                mov track_1_tail, ax

;********************** Update Track 2 Queue ******************************
check_track_2:

                test active_queues,4
                jz check_track_3

                mov cur_track,4
                mov ax,track_2_tail
                mov bx,size queue_ptr
                add bx,ax
                cmp bx,offset dgroup:track_3_base ;at end of array
                jnz not_end_2

                lea bx,track_2_base ;if at end, point to start
not_end_2:
                cmp bx,track_2_head       ;if tail + size = head, queue is full
                jz check_track_3

                lea si,track_2_on  ;point to on and off wait lists
                lea di,track_2_off
                les bx,note_2_ptr         ;and note data

                lea dx, timer_2
                mov timer, dx

                call compute_note         ;with ax = queue pointer
                mov word ptr note_2_ptr,ax  ;update raw note pointer
                mov ax, track_2_tail
                add ax, size queue_ptr

                cmp ax,offset dgroup:track_3_base ;at end of array
                jnz nott_end_2

                lea ax,track_2_base ;if at end, point to start

nott_end_2:
                mov track_2_tail, ax

;********************** Update Track 3 Queue ****************************

check_track_3:

                test active_queues,8
                jz check_track_4

                mov cur_track,8
                mov ax,track_3_tail
                mov bx,size queue_ptr
                add bx,ax
                cmp bx,offset dgroup:track_4_base ;at end of array
                jnz not_end_3

                lea bx,track_3_base ;if at end, point to start

not_end_3:
                cmp bx,track_3_head        ;if tail + size = head, queue is full
                jz check_track_4

                lea si,track_3_on  ;point to on and off wait lists
                lea di,track_3_off
                les bx,note_3_ptr          ;and note data

                lea dx, timer_3
                mov timer, dx

                call compute_note          ;with ax = queue pointer
                mov word ptr note_3_ptr,ax  ;update raw note pointer

                mov ax, track_3_tail
                add ax, size queue_ptr

                cmp ax,offset dgroup:track_4_base ;at end of array
                jnz nott_end_3

                lea ax,track_3_base ;if at end, point to start

nott_end_3:
                mov track_3_tail, ax

;********************** Update Track 4 Queue ****************************

check_track_4:
```

```
                test active_queues,10H
                jz check_track_5

                mov cur_track,10H
                mov ax,track_4_tail
                mov bx,size queue_ptr
                add bx,ax
                cmp bx,offset dgroup:track_5_base ;at end of array
                jnz not_end_4

                lea bx,track_4_base ;if at end, point to start
not_end_4:
                cmp bx,track_4_head         ;if tail + size = head, queue is full
                jz check_track_5

                lea si,track_4_on  ;point to on and off wait lists
                lea di,track_4_off
                les bx,note_4_ptr           ;and note data

                lea dx, timer_4
                mov timer, dx

                call compute_note           ;with ax = queue pointer
                mov word ptr note_4_ptr,ax  ;update raw note pointer

                mov ax, track_4_tail
                add ax, size queue_ptr

                cmp ax,offset dgroup:track_5_base ;at end of array
                jnz nott_end_4

                lea ax,track_4_base ;if at end, point to start
nott_end_4:
                mov track_4_tail, ax

;********************** Update Track 5 Queue *****************************

check_track_5:

                test active_queues,20H
                jz check_track_6

                mov cur_track,20H
                mov ax,track_5_tail
                mov bx,size queue_ptr
                add bx,ax
                cmp bx,offset dgroup:track_6_base ;at end of array
                jnz not_end_5

                lea bx,track_5_base ;if at end, point to start
not_end_5:
                cmp bx,track_5_head         ;if tail + size = head, queue is full
                jz check_track_6
```

```
                 lea si,track_5_on  ;point to on and off wait lists
                 lea di,track_5_off
                 les bx,note_5_ptr         ;and note data

                 lea dx, timer_5
                 mov timer, dx

                 call compute_note         ;with ax = queue pointer
                 mov word ptr note_5_ptr,ax  ;update raw note pointer

                 mov ax, track_5_tail
                 add ax, size queue_ptr

                 cmp ax,offset dgroup:track_6_base ;at end of array
                 jnz nott_end_5

                 lea ax,track_5_base ;if at end, point to start

nott_end_5:
                 mov track_5_tail, ax


;********************** Update Track 6 Queue ****************************

check_track_6:
                 test active_queues,40H
                 jz check_track_7

                 mov cur_track,40H
                 mov ax,track_6_tail
                 mov bx,size queue_ptr
                 add bx,ax
                 cmp bx,offset dgroup:track_7_base ;at end of array
                 jnz not_end_6

                 lea bx,track_6_base ;if at end, point to start

not_end_6:
                 cmp bx,track_6_head       ;if tail + size = head, queue is full
                 jz check_track_7

                 lea si,track_6_on  ;point to on and off wait lists
                 lea di,track_6_off
                 les bx,note_6_ptr         ;and note data

                 lea dx, timer_6
                 mov timer, dx

                 call compute_note         ;with ax = queue pointer
                 mov word ptr note_6_ptr,ax  ;update raw note pointer

                 mov ax, track_6_tail
                 add ax, size queue_ptr

                 cmp ax,offset dgroup:track_7_base ;at end of array
                 jnz nott_end_6
```

```
                lea ax,track_6_base ;if at end, point to start
nott_end_6:
                mov track_6_tail, ax

;********************* Update Track 7 Queue ****************************

check_track_7:
                test active_queues,80H
                jz end_update

                mov cur_track,80H
                mov ax,track_7_tail
                mov bx,size queue_ptr
                add bx,ax
                cmp bx,offset dgroup:last_base ;at end of array
                jnz not_end_7

                lea bx,track_7_base ;if at end, point to start
not_end_7:
                cmp bx,track_7_head       ;if tail + size = head, queue is full
                jz end_update

                lea si,track_7_on  ;point to on and off wait lists
                lea di,track_7_off
                les bx,note_7_ptr          ;and note data

                lea dx, timer_7
                mov timer, dx

                call compute_note          ;with ax = queue pointer
                mov word ptr note_7_ptr,ax  ;update raw note pointer

                mov ax, track_7_tail
                add ax, size queue_ptr

                cmp ax,offset dgroup:last_base ;at end of array
                jnz nott_end_7

                lea ax,track_7_base ;if at end, point to start
nott_end_7:
                mov track_7_tail, ax
end_update:
                cmp queue_updated,0
                jz queues_done
                jmp queue_loop
queues_done:
                pop di
                pop si
                ret

_update_queues endp
```

```
;***************************************************************************
;*                     Send MPU-401 a Command In BL                        *
;***************************************************************************

send_mpu_command proc near

                cli                             ;Prevent DSR from triggering int

                mov dx,STAT_PORT
smc_loop:
                in al,dx
                and al,DRR
                cmp al,DRR                      ;Is MPU 401 ready to receive data
                jz smc_loop

                mov al,bl                       ;Send command
                out dx,al

ack_loop:
                wait_for_dsr

                mov dx,DATA_PORT
                in al,dx
                cmp al,ACK
                jz ack_received

                call process_data               ;It is not acking because it is trying
int
                mov dx,STAT_PORT
                jmp ack_loop

ack_received:
                sti
                ret

send_mpu_command endp

;***************************************************************************
;*                     Send MPU-401 Data Byte In BL                        *
;***************************************************************************

send_mpu_data   proc near

                mov dx,STAT_PORT
smd_loop:
                in al,dx
                and al,DRR
                cmp al,DRR                      ;Is MPU 401 ready to receive data
                jz smd_loop

                mov dx,DATA_PORT
                mov al,bl
                out dx,al

                ret

send_mpu_data   endp
```

```
;****************************************************************************
;*                      Send MPU-401 a Note Pointed By SI                   *
;****************************************************************************

send_note_data proc near

                cmp byte ptr[si].timing_byte,TIMING_OVERFLOW
                jz send_overflow
                cmp byte ptr[si].midi_command,DATA_END
                jz send_data_end

;****************************************************************************
;               GCO Override note
;****************************************************************************

                mov cx,3                        ;timing,note_on,pitch
snd_loop1:
                mov dx,STAT_PORT
drr_loop1:
                in al,dx
                and al,DRR
                cmp al,DRR
                je drr_loop1

                mov dx,DATA_PORT
                lodsb
                out dx,al

                loop snd_loop1

                mov dx,STAT_PORT
drr_loop2:
                in al,dx
                and al,DRR
                cmp al,DRR
                je drr_loop2

                mov dx,DATA_PORT
                lodsb
                cmp al,0
                jz is_zero_vel
                mov al,bl                       ;Velocity override is in BL
is_zero_vel:
                out dx,al

                ret
;****************************************************************************

;****************************************************************************

send_overflow:
                mov dx,STAT_PORT
drr_loop3:
                in al,dx
                and al,DRR
                cmp al,DRR
                je drr_loop3

                mov dx,DATA_PORT
                lodsb
                out dx,al

                add si,3
                ret

send_data_end:
                mov cx,2                        ;timing,data end
data_end_loop:
                mov dx,STAT_PORT
drr_loop4:
                in al,dx
                and al,DRR
                cmp al,DRR
                je drr_loop4

                mov dx,DATA_PORT                   ;and data end message
                lodsb
                out dx,al

                loop data_end_loop

                add si,2
                ret

send_note_data endp


;***********************************************************************
;*                        Some Initializing Stuff
;***********************************************************************


                public _reset_mpu
_reset_mpu      proc near

                mov bl,RESET
                call send_mpu_command

                ret
_reset_mpu      endp


                public _init_mpu
_init_mpu       proc near

                mov bl, SET_TIMEBASE
                call send_mpu_command

                mov bl, CLOCK_TO_HOST_OFF
                call send_mpu_command

                mov bl,SEND_MEASURE_END_OFF
```

```
                    call send_mpu_command

                    mov bl,NO_REAL_TIME_OUT
                    call send_mpu_command

                    ret

_init_mpu           endp


                    public _stop_clock_to_host
_stop_clock_to_host proc near

                    mov bl, CLOCK_TO_HOST_OFF
                    call send_mpu_command

                    ret
_stop_clock_to_host endp


;*****************************************************************************
;*                   Start Playing Notes In The Play Queue                    *
;*****************************************************************************

                    public _start_play
_start_play         proc near

                    mov _song_is_playing, 1

                    mov bl,ACTIVATE_TRACKS
                    call send_mpu_command

                    mov bl, _active_tracks
                    call send_mpu_data

                    mov bl, CLEAR_PLAY_COUNTERS
                    call send_mpu_command

                    mov bl, START_PLAY
                    call send_mpu_command

                    ret

_start_play         endp


                    public _clear_play_counters
_clear_play_counters proc near

                    mov bl, CLEAR_PLAY_COUNTERS
                    call send_mpu_command
                    ret

_clear_play_counters endp

;*****************************************************************************
```

```
;*                               Stop Playing                              *
;***************************************************************************

                public _stop_play
_stop_play      proc near

                mov bl, STOP_PLAY
                call send_mpu_command

                ret

_stop_play      endp

;***************************************************************************
;*                           Set the MPU-401 Tempo                         *
;***************************************************************************

                public _set_tempo
_set_tempo      proc near

                push bp
                mov bp,sp

                mov bl, SET_TEMPO
                call send_mpu_command

                mov bl,[bp+4]
                call send_mpu_data

                pop bp
                ret

_set_tempo      endp

;***************************************************************************
;*              Save Old IRQ 7, Int 0F Vector                              *
;***************************************************************************

                public _stash_int
_stash_int      proc near

                mov al,0FH
                mov ah,35h                  ;code for get vector
                int 21h
                mov word ptr old_int_0F,bx ;stash old vector away
                mov word ptr old_int_0F[2],es

                ret

_stash_int      endp

;***************************************************************************
;*                  Set Int Vector For MPU-401 Operation                   *
;***************************************************************************

                public _set_mpu_int
_set_mpu_int    proc near
```

```
                push ds
                push cs
                pop ds

                mov al,0FH
                mov ah,25h
                mov dx,offset mpu_int
                int 21h

                pop ds

                in al,INT_MASK                  ;Get mask
                mov old_mask,al
                and al,01111111B                ;enable IRQ 7
                out INT_MASK,al                 ;put it back

                ret

_set_mpu_int    endp

;*****************************************************************************
;*              Restore Int 0F Vector To Original State                      *
;*****************************************************************************

                public _fix_int
_fix_int        proc near

                push ds

                mov dx,word ptr old_int_0F
                mov ax,word ptr old_int_0F[2]
                mov ds,ax

                mov al,0FH
                mov ah,25h
                int 21h
                pop ds

                mov al,old_mask
                out INT_MASK,al                 ;put it back

                ret

_fix_int        endp

;*****************************************************************************
;*                      The Int For MPU-401 Operation                        *
;*****************************************************************************

mpu_int         proc near

                push ax
                push bx
                push cx
                push dx
                push si
                push ds
                mov ax,dgroup
                mov ds,ax

                push es

                mov dx,STAT_PORT
                in al,dx
                and al,DSR                      ;Is data ready
                cmp al,DSR
                jnz is_midi                     ;if no data from MPU-401, try old int

                pop es
                pop ds
                pop si
                pop dx
                pop cx
                pop bx
                pop ax
                jmp old_int_0F
is_midi:
                mov dx,DATA_PORT
                in al,dx

                call process_data

                mov al,EOI                      ;reset master interrupt controller
                out INT_CMD,al

                pop es
                pop ds
                pop si
                pop dx
                pop cx
                pop bx
                pop ax
                iret
mpu_int         endp


process_data    proc near

                cmp al, 0EFH                    ;is it a timing byte
                ja is_mpu_message

                jmp end_int
is_mpu_message:
                mov bl,al
                and bl,1111B
                xor bh,bh
                shl bx,1

                jmp mpu_messages[bx]
```

```
track0_data_request:                          ;F0
                  mov si, track_0_head
                  mov bl, byte ptr _velocity_override_0
                  call send_note_data
                  cmp si, offset dgroup:track_1_base
                  jl track_0_end
                  lea si,track_0_base
track_0_end:
                  mov track_0_head,si
                  jmp end_int

track1_data_request:                          ;F1
                  mov si, track_1_head
                  mov bl, byte ptr _velocity_override_1
                  call send_note_data
                  cmp si, offset dgroup:track_2_base
                  jl track_1_end
                  lea si,track_1_base
track_1_end:
                  mov track_1_head,si
                  jmp end_int

track2_data_request:                          ;F2
                  mov si, track_2_head
                  mov bl, byte ptr _velocity_override_2
                  call send_note_data
                  cmp si, offset dgroup:track_3_base
                  jl track_2_end
                  lea si,track_2_base
track_2_end:
                  mov track_2_head,si
                  jmp end_int

track3_data_request:                          ;F3
                  mov si, track_3_head
                  mov bl, byte ptr _velocity_override_3
                  call send_note_data
                  cmp si, offset dgroup:track_4_base
                  jl track_3_end
                  lea si,track_3_base
track_3_end:
                  mov track_3_head,si
                  jmp end_int

track4_data_request:                          ;F4
                  mov si, track_4_head
                  mov bl, byte ptr _velocity_override_4
                  call send_note_data
                  cmp si, offset dgroup:track_5_base
                  jl track_4_end
                  lea si,track_4_base
track_4_end:
                  mov track_4_head,si
                  jmp end_int

track5_data_request:                          ;F5
                  mov si, track_5_head
```

```
                    mov bl, byte ptr _velocity_override_5
                    call send_note_data
                    cmp si, offset dgroup:track_6_base
                    jl track_5_end
                    lea si,track_5_base
track_5_end:
                    mov track_5_head,si
                    jmp end_int

track6_data_request:                           ;F6
                    mov si, track_6_head
                    mov bl, byte ptr _velocity_override_6
                    call send_note_data
                    cmp si, offset dgroup:track_7_base
                    jl track_6_end
                    lea si, track_6_base
track_6_end:
                    mov track_6_head,si
                    jmp end_int

track7_data_request:                           ;F7
                    mov si, track_7_head
                    mov bl, byte ptr _velocity_override_7
                    call send_note_data
                    cmp si, offset dgroup:last_base ;at end of array
                    jl track_7_end
                    lea si,track_7_base
track_7_end:
                    mov track_7_head,si
                    jmp end_int

timing_data_overflow:                          ;F8
                    jmp end_int

conductor_data_request:                        ;F9
                    jmp end_int

undefined1:                                    ;FA
                    jmp end_int

undefined2:                                    ;FB
                    jmp end_int

all_end:                                       ;FC
                    mov _song_is_playing,0
                    jmp end_int

clock_to_host:                                 ;FD
                    inc _message_ready
                    jmp end_int

is_ack:                                        ;FE
                    jmp end_int

system_message:                                ;FF
                    jmp end_int

song_position:
```

```
                jmp end_int
midi_start:
                jmp end_int
midi_stop:
                jmp end_int
midi_continue:
                jmp end_int
system_exclusive:
                jmp end_int
not_used:
                jmp end_int
end_int:
                ret
process_data    endp

;*****************************************************************************
;*                            Set the MPU-401 Clock To Host
;*****************************************************************************

                public _set_clock_to_host
_set_clock_to_host proc near

                push bp
                mov bp,sp

                mov bl, SET_CLOCK_TO_HOST
                call send_mpu_command

                mov bl, [bp+4]
                call send_mpu_data

                mov bl, CLOCK_TO_HOST_ON
                call send_mpu_command

                pop bp
                ret

_set_clock_to_host  endp

                end
```

```
;
;________________________________________________________________________
;
;                    9 x 16 Character Generator And
;                 Screen Control Functions SCREEN.ASM
;                     Hercules Monochrome Graphics
;
;                      WDI Guest Controlled Orchestra
;
;
;
;________________________________________________________________________
;

INDEX_6845      equ 3B4H
CONTROL_6845    equ 3B8H
NO_DOTS         equ 00000000B
ALL_DOTS        equ 11111111B

                .model small
                .data

chars db  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H
      db  00H, 00H, 7EH, 81H,0A5H, 81H, 81H,0BDH, 99H, 81H, 7EH, 00H, 00H, 00H
      db  00H, 00H, 7EH,0FFH,0DBH,0FFH,0FFH,0C3H,0E7H,0FFH, 7EH, 00H, 00H, 00H
      db  00H, 00H, 00H, 36H, 7FH, 7FH, 7FH, 7FH, 3EH, 1CH, 08H, 00H, 00H, 00H
      db  00H, 00H, 00H, 08H, 1CH, 3EH, 7FH, 3EH, 1CH, 08H, 00H, 00H, 00H, 00H
      db  00H, 00H, 18H, 3CH, 3CH,0E7H,0E7H,0E7H, 18H, 18H, 3CH, 00H, 00H, 00H
      db  00H, 00H, 18H, 3CH, 7EH,0FFH,0FFH, 7EH, 18H, 18H, 3CH, 00H, 00H, 00H
      db  00H, 00H, 00H, 00H, 00H, 18H, 3CH, 3CH, 18H, 00H, 00H, 00H, 00H, 00H
      db 0FFH,0FFH,0FFH,0FFH,0FFH,0E7H,0C3H,0C3H,0E7H,0FFH,0FFH,0FFH,0FFH,0FFH
      db  00H, 00H, 00H, 00H, 3CH, 66H, 42H, 42H, 66H, 3CH, 00H, 00H, 00H, 00H
      db 0FFH,0FFH,0FFH,0FFH,0C3H, 99H,0BDH,0BDH, 99H,0C3H,0FFH,0FFH,0FFH,0FFH
      db  00H, 00H, 0FH, 07H, 0DH, 19H, 3CH, 66H, 66H, 66H, 3CH, 00H, 00H, 00H
      db  00H, 00H, 3CH, 66H, 66H, 66H, 3CH, 18H, 7EH, 18H, 18H, 00H, 00H, 00H
      db  00H, 00H, 3FH, 33H, 3FH, 30H, 30H, 30H, 70H,0F0H,0E0H, 00H, 00H, 00H
      db  00H, 00H, 7FH, 63H, 7FH, 63H, 63H, 63H, 67H,0E7H,0E6H,0C0H, 00H, 00H
      db  00H, 00H, 18H, 18H,0DBH, 3CH,0E7H, 3CH,0DBH, 18H, 18H, 00H, 00H, 00H

      db  00H, 00H, 40H, 60H, 70H, 7CH, 7FH, 7CH, 70H, 60H, 40H, 00H, 00H, 00H
;10h
      db  00H, 00H, 01H, 03H, 07H, 1FH, 7FH, 1FH, 07H, 03H, 01H, 00H, 00H, 00H
      db  00H, 00H, 18H, 3CH, 7EH, 18H, 18H, 18H, 7EH, 3CH, 18H, 00H, 00H, 00H
      db  00H, 00H, 33H, 33H, 33H, 33H, 33H, 33H, 00H, 33H, 33H, 00H, 00H, 00H
      db  00H, 00H, 7FH,0DBH,0DBH,0DBH, 7BH, 1BH, 1BH, 1BH, 1BH, 00H, 00H, 00H
      db  00H, 3EH, 63H, 30H, 1CH, 36H, 63H, 63H, 36H, 1CH, 06H, 63H, 3EH, 00H
      db  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 7FH, 7FH, 7FH, 00H, 00H, 00H
      db  00H, 00H, 18H, 3CH, 7EH, 18H, 18H, 18H, 7EH, 3CH, 18H, 7EH, 00H, 00H
      db  00H, 00H, 18H, 3CH, 7EH, 18H, 18H, 18H, 18H, 18H, 18H, 00H, 00H, 00H
      db  00H, 00H, 18H, 18H, 18H, 18H, 18H, 18H, 7EH, 3CH, 18H, 00H, 00H, 00H
      db  00H, 00H, 00H, 00H, 0CH, 06H, 7FH, 06H, 0CH, 00H, 00H, 00H, 00H, 00H
      db  00H, 00H, 00H, 00H, 18H, 30H, 7FH, 30H, 18H, 00H, 00H, 00H, 00H, 00H
      db  00H, 00H, 00H, 00H, 00H, 60H, 60H, 60H, 7FH, 00H, 00H, 00H, 00H, 00H
      db  00H, 00H, 00H, 00H, 24H, 66H,0FFH, 66H, 24H, 00H, 00H, 00H, 00H, 00H
      db  00H, 00H, 00H, 08H, 1CH, 1CH, 3EH, 3EH, 7FH, 7FH, 00H, 00H, 00H, 00H
      db  00H, 00H, 00H, 7FH, 7FH, 3EH, 3EH, 1CH, 1CH, 08H, 00H, 00H, 00H, 00H

      db  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H
      db  00H, 00H, 18H, 3CH, 3CH, 3CH, 18H, 18H, 00H, 18H, 18H, 00H, 00H, 00H
db  00H, 63H, 63H, 63H, 22H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H
db  00H, 00H, 36H, 36H, 7FH, 36H, 36H, 36H, 7FH, 36H, 36H, 00H, 00H, 00H
db  0CH, 0CH, 3EH, 63H, 61H, 60H, 3EH, 03H, 43H, 63H, 3EH, 0CH, 0CH, 00H
db  00H, 00H, 00H, 00H, 61H, 63H, 06H, 0CH, 18H, 33H, 63H, 00H, 00H, 00H
db  00H, 00H, 1CH, 36H, 36H, 1CH, 3BH, 6EH, 66H, 66H, 3BH, 00H, 00H, 00H
db  00H, 30H, 30H, 30H, 60H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H
db  00H, 00H, 0CH, 18H, 30H, 30H, 30H, 30H, 30H, 18H, 0CH, 00H, 00H, 00H
db  00H, 00H, 18H, 0CH, 06H, 06H, 06H, 06H, 06H, 0CH, 18H, 00H, 00H, 00H
db  00H, 00H, 00H, 00H, 66H, 3CH,0FFH, 3CH, 66H, 00H, 00H, 00H, 00H, 00H
db  00H, 00H, 00H, 18H, 18H, 18H,0FFH, 18H, 18H, 18H, 00H, 00H, 00H, 00H
db  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 18H, 18H, 18H, 30H, 00H, 00H
db  00H, 00H, 00H, 00H, 00H, 00H,0FFH, 00H, 00H, 00H, 00H, 00H, 00H, 00H
db  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 18H, 18H, 00H, 00H, 00H
db  00H, 00H, 01H, 03H, 06H, 0CH, 18H, 30H, 60H, 40H, 00H, 00H, 00H, 00H

db  00H, 00H, 3EH, 63H, 67H, 6FH, 7BH, 73H, 63H, 63H, 3EH, 00H, 00H, 00H
db  00H, 00H, 0CH, 1CH, 3CH, 0CH, 0CH, 0CH, 0CH, 0CH, 3FH, 00H, 00H, 00H
db  00H, 00H, 3EH, 63H, 03H, 06H, 0CH, 18H, 30H, 63H, 7FH, 00H, 00H, 00H
db  00H, 00H, 3EH, 63H, 03H, 03H, 1EH, 03H, 03H, 63H, 3EH, 00H, 00H, 00H
db  00H, 00H, 06H, 0EH, 1EH, 36H, 66H, 7FH, 06H, 06H, 0FH, 00H, 00H, 00H
db  00H, 00H, 7FH, 60H, 60H, 60H, 7EH, 03H, 03H, 63H, 3EH, 00H, 00H, 00H
db  00H, 00H, 1CH, 30H, 60H, 60H, 7EH, 63H, 63H, 63H, 3EH, 00H, 00H, 00H
db  00H, 00H, 7FH, 63H, 03H, 06H, 0CH, 18H, 18H, 18H, 18H, 00H, 00H, 00H
db  00H, 00H, 3EH, 63H, 63H, 63H, 3EH, 63H, 63H, 63H, 3EH, 00H, 00H, 00H
db  00H, 00H, 3EH, 63H, 63H, 63H, 3FH, 03H, 03H, 06H, 3CH, 00H, 00H, 00H
db  00H, 00H, 00H, 18H, 18H, 00H, 00H, 00H, 18H, 18H, 00H, 00H, 00H, 00H
db  00H, 00H, 00H, 18H, 18H, 00H, 00H, 00H, 18H, 18H, 30H, 00H, 00H, 00H
db  00H, 00H, 06H, 0CH, 18H, 30H, 60H, 30H, 18H, 0CH, 06H, 00H, 00H, 00H
db  00H, 00H, 00H, 00H, 00H, 7EH, 00H, 00H, 7EH, 00H, 00H, 00H, 00H, 00H
db  00H, 00H, 60H, 30H, 18H, 0CH, 06H, 0CH, 18H, 30H, 60H, 00H, 00H, 00H
db  00H, 00H, 3EH, 63H, 63H, 06H, 0CH, 0CH, 00H, 0CH, 0CH, 00H, 00H, 00H

db  00H, 00H, 3EH, 63H, 63H, 6FH, 6FH, 6FH, 6EH, 60H, 3EH, 00H, 00H, 00H  ;
db  00H, 00H, 08H, 1CH, 36H, 63H, 63H, 7FH, 63H, 63H, 63H, 00H, 00H, 00H
db  00H, 00H, 7EH, 33H, 33H, 33H, 3EH, 33H, 33H, 33H, 7EH, 00H, 00H, 00H
db  00H, 00H, 1EH, 33H, 61H, 60H, 60H, 60H, 61H, 33H, 1EH, 00H, 00H, 00H
db  00H, 00H, 7CH, 36H, 33H, 33H, 33H, 33H, 33H, 36H, 7CH, 00H, 00H, 00H
db  00H, 00H, 7FH, 33H, 31H, 34H, 3CH, 34H, 31H, 33H, 7FH, 00H, 00H, 00H
db  00H, 00H, 7FH, 33H, 31H, 34H, 3CH, 34H, 30H, 30H, 78H, 00H, 00H, 00H
db  00H, 00H, 1EH, 33H, 61H, 60H, 60H, 6FH, 63H, 33H, 1DH, 00H, 00H, 00H
db  00H, 00H, 63H, 63H, 63H, 63H, 7FH, 63H, 63H, 63H, 63H, 00H, 00H, 00H
db  00H, 00H, 3CH, 18H, 18H, 18H, 18H, 18H, 18H, 18H, 3CH, 00H, 00H, 00H
db  00H, 00H, 0FH, 06H, 06H, 06H, 06H, 06H, 66H, 66H, 3CH, 00H, 00H, 00H
db  00H, 00H, 73H, 33H, 36H, 36H, 3CH, 36H, 36H, 33H, 73H, 00H, 00H, 00H
db  00H, 00H, 78H, 30H, 30H, 30H, 30H, 30H, 31H, 33H, 7FH, 00H, 00H, 00H
db  00H, 00H,0C3H,0E7H,0FFH,0DBH,0C3H,0C3H,0C3H,0C3H,0C3H, 00H, 00H, 00H
db  00H, 00H, 63H, 73H, 7BH, 7FH, 6FH, 67H, 63H, 63H, 63H, 00H, 00H, 00H
db  00H, 00H, 1CH, 36H, 63H, 63H, 63H, 63H, 63H, 36H, 1CH, 00H, 00H, 00H

db  00H, 00H, 7EH, 33H, 33H, 33H, 3EH, 30H, 30H, 30H, 78H, 00H, 00H, 00H ;5
db  00H, 00H, 3EH, 63H, 63H, 63H, 63H, 6BH, 6FH, 3EH, 06H, 07H, 00H, 00H
db  00H, 00H, 7EH, 33H, 33H, 33H, 3EH, 36H, 33H, 33H, 73H, 00H, 00H, 00H
db  00H, 00H, 3EH, 63H, 63H, 30H, 1CH, 06H, 63H, 63H, 3EH, 00H, 00H, 00H
db  00H, 00H,0FFH,0DBH, 99H, 18H, 18H, 18H, 18H, 18H, 3CH, 00H, 00H, 00H
db  00H, 00H, 63H, 63H, 63H, 63H, 63H, 63H, 63H, 63H, 3EH, 00H, 00H, 00H
db  00H, 00H,0C3H,0C3H,0C3H,0C3H,0C3H,0C3H, 66H, 3CH, 18H, 00H, 00H, 00H
db  00H, 00H,0C3H,0C3H,0C3H,0C3H,0DBH,0DBH,0FFH, 66H, 66H, 00H, 00H, 00H
db  00H, 00H,0C3H,0C3H, 66H, 3CH, 18H, 3CH, 66H,0C3H,0C3H, 00H, 00H, 00H
```

```
     db  00H, 00H,0C3H,0C3H,0C3H, 66H, 3CH, 18H, 18H, 18H, 3CH, 00H, 00H, 00H
     db  00H, 00H,0FFH,0C3H, 86H, 0CH, 18H, 30H, 61H,0C3H,0FFH, 00H, 00H, 00H
     db  00H, 00H, 3CH, 30H, 30H, 30H, 30H, 30H, 30H, 30H, 3CH, 00H, 00H, 00H
     db  00H, 00H, 40H, 60H, 70H, 38H, 1CH, 0EH, 07H, 03H, 01H, 00H, 00H, 00H
     db  00H, 00H, 3CH, 0CH, 0CH, 0CH, 0CH, 0CH, 0CH, 0CH, 3CH, 00H, 00H, 00H
     db  08H, 1CH, 36H, 63H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H
     db  00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H,0FFH, 00H

     db  18H, 18H, 0CH, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H ;t
     db  00H, 00H, 00H, 00H, 00H, 3CH, 06H, 3EH, 66H, 66H, 3BH, 00H, 00H, 00H
     db  00H, 00H, 70H, 30H, 30H, 3CH, 36H, 33H, 33H, 33H, 6EH, 00H, 00H, 00H
     db  00H, 00H, 00H, 00H, 00H, 3EH, 63H, 60H, 60H, 63H, 3EH, 00H, 00H, 00H
     db  00H, 00H, 0EH, 06H, 06H, 1EH, 36H, 66H, 66H, 66H, 3BH, 00H, 00H, 00H
     db  00H, 00H, 00H, 00H, 00H, 3EH, 63H, 7FH, 60H, 63H, 3EH, 00H, 00H, 00H
     db  00H, 00H, 1CH, 36H, 32H, 30H, 7CH, 30H, 30H, 30H, 78H, 00H, 00H, 00H
     db  00H, 00H, 00H, 00H, 00H, 3BH, 66H, 66H, 66H, 3EH, 06H, 66H, 3CH, 00H
     db  00H, 00H, 70H, 30H, 30H, 36H, 3BH, 33H, 33H, 33H, 73H, 00H, 00H, 00H
     db  00H, 00H, 0CH, 0CH, 00H, 1CH, 0CH, 0CH, 0CH, 0CH, 1EH, 00H, 00H, 00H
     db  00H, 00H, 06H, 06H, 00H, 0EH, 06H, 06H, 06H, 06H, 66H, 66H, 3CH, 00H
     db  00H, 00H, 70H, 30H, 30H, 33H, 36H, 3CH, 36H, 33H, 73H, 00H, 00H, 00H
     db  00H, 00H, 1CH, 0CH, 0CH, 0CH, 0CH, 0CH, 0CH, 0CH, 1EH, 00H, 00H, 00H
     db  00H, 00H, 00H, 00H, 00H,0E6H,0FFH,0DBH,0DBH,0DBH,0DBH, 00H, 00H, 00H
     db  00H, 00H, 00H, 00H, 00H, 6EH, 33H, 33H, 33H, 33H, 33H, 00H, 00H, 00H
     db  00H, 00H, 00H, 00H, 00H, 3EH, 63H, 63H, 63H, 63H, 3EH, 00H, 00H, 00H

     db  00H, 00H, 00H, 00H, 00H, 6EH, 33H, 33H, 33H, 3EH, 30H, 30H, 78H, 00H ;-
     db  00H, 00H, 00H, 00H, 00H, 3BH, 66H, 66H, 66H, 3EH, 06H, 06H, 0FH, 00H
     db  00H, 00H, 00H, 00H, 00H, 6EH, 3BH, 33H, 30H, 30H, 78H, 00H, 00H, 00H
     db  00H, 00H, 00H, 00H, 00H, 3EH, 63H, 38H, 0EH, 63H, 3EH, 00H, 00H, 00H
     db  00H, 00H, 08H, 18H, 18H, 7EH, 18H, 18H, 18H, 1BH, 0EH, 00H, 00H, 00H
     db  00H, 00H, 00H, 00H, 00H, 66H, 66H, 66H, 66H, 66H, 3BH, 00H, 00H, 00H
     db  00H, 00H, 00H, 00H, 00H,0C3H,0C3H,0C3H, 66H, 3CH, 18H, 00H, 00H, 00H
     db  00H, 00H, 00H, 00H, 00H,0C3H,0C3H,0DBH,0DBH,0FFH, 66H, 00H, 00H, 00H
     db  00H, 00H, 00H, 00H, 00H, 63H, 36H, 1CH, 1CH, 36H, 63H, 00H, 00H, 00H
     db  00H, 00H, 00H, 00H, 00H, 63H, 63H, 63H, 63H, 3FH, 03H, 06H, 3CH, 00H
     db  00H, 00H, 00H, 00H, 00H, 7FH, 66H, 0CH, 18H, 33H, 7FH, 00H, 00H, 00H
     db  00H, 00H, 0EH, 18H, 18H, 18H, 70H, 18H, 18H, 18H, 0EH, 00H, 00H, 00H
     db  00H, 00H, 18H, 18H, 18H, 18H, 00H, 18H, 18H, 18H, 18H, 00H, 00H, 00H
     db  00H, 00H, 70H, 18H, 18H, 18H, 0EH, 18H, 18H, 18H, 70H, 00H, 00H, 00H
     db  00H, 00H, 3BH, 6EH, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H, 00H
     db  00H, 00H, 00H, 00H, 08H, 1CH, 36H, 63H, 63H, 7FH, 00H, 00H, 00H, 00H

herc_screen     dw 0B000H

gtable          db 35H,2DH,2EH,07H
                db 5BH,02H,57H,57H
                db 02H,03H,00H,00H

ttable          db 61H,50H,52H,0FH
                db 19H,06H,19H,19H
                db 02H,0DH,0BH,0CH
                public _init_herc

forty_hex       dw 40H

                .code
```

```
old_mode        db ?

                public eight
eight           db 8
nine            db 9
fourteen        db 14
three_fifteen   dw 315

;*********************************************************************************
;*********************************************************************************

                public _clear_graphics
_clear_graphics proc near

                push di

                mov cx,4000h             ; words to clear
                mov es,herc_screen
                xor di,di
                mov ax,0
                rep stosw                ; clear mem

                pop di
                ret

_clear_graphics endp

;*********************************************************************************
;*********************************************************************************

;      display_text(string, x, y);
;                      4    6  8
;*********************************************************************************
;*********************************************************************************


                public _display_text
_display_text   proc near

                push bp
                mov bp,sp

                mov si,[bp+4]
                mov ax,0B000H
                mov es,ax
                mov bx,offset chars

                mov ax,[bp+8]            ;Get line
                and ax,1
                jz even_line
                jmp odd_line

even_line:
                mov ax,[bp+8]            ;Get line
                mul three_fifteen
                mov di,ax
                mov ax,[bp+6]            ;Get col
                add di,ax
                div eight
                mov cl,ah               ; Use remainder for shifts
                xor ah,ah
                add di,ax

c_loop:
                mov al,[si]
                cmp al,0
                jnz c_ok
                jmp c_quit
c_ok:
                push bx
                inc si
                mul fourteen
                add bx,ax
                xor ah,ah

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+2000H],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+4000H],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+6000H],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+5ah],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+205aH],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+405aH],ax
                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+605aH],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+0b4h],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+20b4H],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+40b4H],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+60b4H],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+10eH],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+210eH],ax

                pop bx
                inc cl
                cmp cl,8
                jl not_8
                xor cl,cl
                inc di
not_8:
                inc di
                jmp c_loop

c_quit:
                pop bp
                ret
odd_line:
                mov ax,[bp+8]           ;Get line
                mul three_fifteen
                sub ax,45
                mov di,ax
                mov ax,[bp+6]          ;Get col
                add di,ax
                div eight
                mov cl,ah              ; Use remainder for shifts
                xor ah,ah
                add di,ax

oc_loop:
                mov al,[si]
                cmp al,0
                jnz oc_ok
                jmp oc_quit
oc_ok:
                push bx
                inc si
                mul fourteen
                add bx,ax
                xor ah,ah

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+4000H],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+6000H],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+5ah],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+205aH],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+405aH],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+605aH],ax
                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+0b4h],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+20b4H],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+40b4H],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+60b4H],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+10eH],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+210eH],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+410eH],ax

                mov al,[bx]
                xor ah,ah
                ror ax,cl
                inc bx
                xor es:[di+610eH],ax

                pop bx
                inc cl
                cmp cl,8
                jl onot_8
                xor cl,cl
                inc di
onot_8:
                inc di
                jmp oc_loop
```

```
oc_quit:
                pop bp
                ret

_display_text  endp

;*******************************************************************************
;*******************************************************************************

;          display_blanks(count, x, y);
;                          4     6  8

;*******************************************************************************
;*******************************************************************************

                public _display_blanks
_display_blanks proc near

                push bp
                mov bp,sp
                push ds

                mov ax,0B000H
                mov ds,ax

                mov ax,[bp+8]           ;Get line
                test ax,1
                jz even_b_line
                jmp odd_b_line

even_b_line:
                mul three_fifteen
                mov di,ax
                mov ax,[bp+6]           ;Get col
                add di,ax
                div eight
                mov cl,ah                ; Use remainder for shifts
                xor ah,ah
                xor ch,ch
                add di,ax

                mov bl,ALL_DOTS          ; Load all dots
                shr bl,cl                ; shift for proper first dot pos
                not bl                   ; Invert

                mov ax,[bp+4]            ; Load count of spaces
                mul nine                 ; Times 9 for dots
                sub ax,8
                add ax,cx                ; Minus (8-shifts)
                div eight                ; Divided by 8 for bytes

                mov cl,al                ; Load counter

                call clear_one           ; Clear first partial byte

                jcxz no_loop
                mov bl, NO_DOTS
```

```
clear_loop:                                 ; Clear all Full bytes
                call clear_one
                loop clear_loop
no_loop:
                mov cl,ah                   ; Get remainder
                mov bl,ALL_DOTS
                shr bl,cl                   ; Finish off remainder of line
                call clear_one

                pop ds
                pop bp
                ret
odd_b_line:
                mul three_fifteen
                sub ax,45
                mov di,ax
                mov ax,[bp+6]               ;Get col
                add di,ax
                div eight
                mov cl,ah                   ; Use remainder for shifts
                xor ah,ah
                xor ch,ch
                add di,ax

                mov bl,ALL_DOTS             ; Load all dots
                shr bl,cl                   ; shift for proper first dot pos
                not bl                      ; Invert

                mov ax,[bp+4]               ; Load count of spaces
                mul nine                    ; Times 9 for dots
                sub ax,8
                add ax,cx                   ; Minus (8-shifts)
                div eight                   ; Divided by 8 for bytes

                mov cl,al                   ; Load counter

                call clear_o_one            ; Clear first partial byte

                jcxz no_o_loop
                mov bl, NO_DOTS
clear_o_loop:                               ; Clear all Full bytes
                call clear_o_one
                loop clear_o_loop
no_o_loop:
                mov cl,ah                   ; Get remainder
                mov bl,ALL_DOTS
                shr bl,cl                   ; Finish off remainder of line
                call clear_o_one

                pop ds
                pop bp
                ret

_display_blanks endp

;*****************************************************************************
;*****************************************************************************
```

```
clear_one       proc near

                and [di],bl
                and [di+2000H],bl
                and [di+4000H],bl
                and [di+6000H],bl
                and [di+5ah],bl
                and [di+205aH],bl
                and [di+405aH],bl
                and [di+605aH],bl
                and [di+0b4h],bl
                and [di+20b4H],bl
                and [di+40b4H],bl
                and [di+60b4H],bl
                and [di+10eH],bl
                and [di+210eH],bl
                inc di

                ret
clear_one       endp

;**************************************************************************
;**************************************************************************

clear_o_one     proc near

                and [di+4000H],bl
                and [di+6000H],bl
                and [di+5ah],bl
                and [di+205aH],bl
                and [di+405aH],bl
                and [di+605aH],bl
                and [di+0b4h],bl
                and [di+20b4H],bl
                and [di+40b4H],bl
                and [di+60b4H],bl
                and [di+10eH],bl
                and [di+210eH],bl
                and [di+410eH],bl
                and [di+610eH],bl

                inc di

                ret

clear_o_one     endp

_init_herc      proc near

                mov dx,3BFH
                mov al,1
                out dx,al
                ret

_init_herc      endp
```

```
;*********************************************************************
;*              SET HERCULES MONOCHROME GRAPHICS MODE               *
;*********************************************************************

                public _set_graphics
_set_graphics   proc near

                mov cx,2000h            ; words to clear
                mov es,herc_screen
                xor di,di
                mov ax,720H
                rep stosw               ; clear mem

                lea si,gtable

                mov dx,INDEX_6845       ; 6845 index port
                mov cx,12               ; 12 params
                xor ah,ah               ; starting from register zero

g_loop:
                mov al,ah
                out dx,al               ; Select register
                inc dx                  ; Point to data port
                lodsb                   ; Get data
                out dx,al               ; Output it to 6845
                inc ah                  ; Next register
                dec dx                  ; Point back to index port
                loop g_loop

                mov dx,CONTROL_6845     ; set graphics
                mov al,00000010b
                out dx,al

                mov cx,4000h            ; words to clear
                mov es,herc_screen
                xor di,di
                mov ax,0
                rep stosw               ; clear mem

                mov cx,3000h
timer:
                aam
                loop timer

                mov dx,CONTROL_6845       ; turn screen on
                mov al,00001010b
                out dx,al

                push ds
                mov ds,forty_hex
                mov si,49H
                mov al,[si]
                mov old_mode,al
                mov byte ptr[si],6
                pop ds

                ret
```

```
_set_graphics  endp

;*********************************************************************
;*                    RESTORE MONOCHROME TEXT MODE                   *
;*********************************************************************

                public _set_text
_set_text       proc near

                mov cx,4000h             ; words to clear
                mov es,herc_screen
                xor di,di
                mov ax,0
                rep stosw                ; clear mem

                lea si,ttable

                mov dx,INDEX_6845        ; 6845 index port
                mov cx,12                ; 12 params
                xor ah,ah                ; starting from register zero

t_loop:
                mov al,ah
                out dx,al                ; Select register
                inc dx                   ; Point to data port
                lodsb                    ; Get data
                out dx,al                ; Output it to 6845
                inc ah                   ; Next register
                dec dx                   ; Point back to index port
                loop t_loop

                mov dx,CONTROL_6845      ; set text mode
                mov al,00000000b
                out dx,al

                mov cx,2000h             ; words to clear
                mov es,herc_screen
                xor di,di
                mov ax,720H
                rep stosw                ; clear mem

                mov cx,3000h
timer1:
                aam
                loop timer1

                mov dx,CONTROL_6845      ; turn screen on
                mov al,00101000b
                out dx,al

                push ds
                mov ds,forty_hex
                mov si,49H
                mov al,old_mode
                mov [si],al
                pop ds

                ret
```

```
_set_text       endp
                end
```

```
;
;                 Low Level Image Processing Routines VIDEO.ASM
;                         Regular smart centroid
;___________________________________________________________________________
;
;                       WDI Guest Controlled Orchestra
;
;
;___________________________________________________________________________
;
;
;        ....xxxxxx......

                 .model small
                 .data

 ; Set Bank 0, 485 Lines, Start at Line 0, External Phase Lock,
 ; Enable Image Acquisition

ACQUIRE_FLAG_ON       equ 00110000B

LOW_CSR_PORT          equ 2F0H
HIGH_CSR_PORT         equ 2F1H
LUT_ADDRESS_PORT      equ 2F2H
RED_LUT_DATA_PORT     equ 2F3H
BLUE_LUT_DATA_PORT    equ 2F4H
GREEN_LUT_DATA_PORT   equ 2F5H
INPUT_LUT_DATA_PORT   equ 2F6H


X_INC            equ 4
Y_INC            equ 6 * 512
X_COUNT          equ 40
Y_COUNT          equ 50

INSET            equ 25        ;25
FIRST_LINE       equ 50
UPPER_LEFT       equ 512 * FIRST_LINE + INSET
UPPER_RIGHT      equ 512 * FIRST_LINE + (384 - INSET)

ZONE_SIZE        equ 10
MIN_COUNT        equ 2


CORR_TEMPLATE_SIZE  equ 30
CORR_HISTORY_SIZE   equ 200


SQUARES_BUFFER_SIZE equ CORR_TEMPLATE_SIZE * CORR_HISTORY_SIZE
WAVE_HISTORY_END    equ offset wave_history_buffer + CORR_HISTORY_SIZE * 2
CORR_SQUARES_END    equ offset corr_squares_buffer + SQUARES_BUFFER_SIZE * 2
CORR_SUMS_END       equ offset _corr_sums_buffer + (CORR_HISTORY_SIZE - 1) * 4

BANDGAP         equ 512
DECISION_ZONE   equ 512
MAX_GOOD_CORR   equ 2048

INITIAL_BANK    equ 0
```

```
HIGH_THRESHOLD equ 0F0h
LOW_THRESHOLD  equ 010h

COUNT_FILT_SHIFT   equ 2                  ;originally 3 was 2
COUNT_FILT_SIZE    equ 4                  ;originally 8 was 4

CENT_FILT_SHIFT    equ 1                  ;originally 2 was 1
CENT_FILT_SIZE     equ 2                  ;originally 4 was 2

inc_y           macro
                local bank_ok
                add si,Y_INC
                jnc bank_ok

                inc bank_number
                mov al,ACQUIRE_FLAG_ON
                add al,bank_number
                mov dx,LOW_CSR_PORT
                out dx,al
bank_ok:
                endm


set_bank        macro
                mov al,ACQUIRE_FLAG_ON
                add al,bank_number
                mov dx,LOW_CSR_PORT
                out dx,al
                endm

get_a_sum       macro
                local clamp_it
                local no_clamp
                mov ax,[di]
                cmp word ptr[di+2],0
                jnz clamp_it

                cmp ax,1000H
                jbe no_clamp

clamp_it:
                mov ax,1000H

no_clamp:
                sub di,4
                endm


ivg_seg         dw 0A000h

herc_screen     dw 0B000H

l_count_filt_sum    dw 0
l_count_filt_buffer dw COUNT_FILT_SIZE dup(0)
l_count_filt_ptr    dw 0

l_x_filt_sum        dw 0
```

```
l_x_filt_buffer     dw CENT_FILT_SIZE dup(0)
l_x_filt_ptr        dw 0

l_y_filt_sum        dw 0
l_y_filt_buffer     dw CENT_FILT_SIZE dup(0)
l_y_filt_ptr        dw 0


r_count_filt_sum    dw 0
r_count_filt_buffer dw COUNT_FILT_SIZE dup(0)
r_count_filt_ptr    dw 0

r_x_filt_sum        dw 0
r_x_filt_buffer     dw CENT_FILT_SIZE dup(0)
r_x_filt_ptr        dw 0

r_y_filt_sum        dw 0
r_y_filt_buffer     dw CENT_FILT_SIZE dup(0)
r_y_filt_ptr        dw 0
                dw 3969, 3844, 3721, 3600, 3481, 3364, 3249
                dw 3136, 3025, 2916, 2809, 2704, 2601, 2500, 2401
                dw 2304, 2209, 2116, 2025, 1936, 1849, 1764, 1681
                dw 1600, 1521, 1444, 1369, 1296, 1225, 1156, 1089
                dw 1024,  961,  900,  841,  784,  729,  676,  625
                dw  576,  529,  484,  441,  400,  361,  324,  289
                dw  256,  225,  196,  169,  144,  121,  100,   81
                dw   64,   49,   36,   25,   16,    9,    4,    1
squares         dw    0
                dw    1,    4,    9,   16,   25,   36,   49,   64
                dw   81,  100,  121,  144,  169,  196,  225,  256
                dw  289,  324,  361,  400,  441,  484,  529,  576
                dw  625,  676,  729,  784,  841,  900,  961, 1024
                dw 1089, 1156, 1225, 1296, 1369, 1444, 1521, 1600
                dw 1681, 1764, 1849, 1936, 2025, 2116, 2209, 2304
                dw 2401, 2500, 2601, 2704, 2809, 2916, 3025, 3136
                dw 3249, 3364, 3481, 3600, 3721, 3844, 3969


corr_squares_ptr    dw corr_squares_buffer
wave_history_ptr    dw wave_history_buffer

                .data?

                public _corr_sums_buffer
_corr_sums_buffer   dd CORR_HISTORY_SIZE dup(?)

corr_squares_buffer dw SQUARES_BUFFER_SIZE dup(?)
wave_history_buffer dw CORR_HISTORY_SIZE dup(?)

corr_count      dw ?

old_graph_0     dw 720*2 dup(?)
old_ax_0        dw ?
old_di_0        dw ?

old_graph_1     dw 720*2 dup(?)
old_ax_1        dw ?
```

```
old_di_1          dw ?

old_graph_2       dw 720*2 dup(?)
old_ax_2          dw ?
old_di_2          dw ?

old_graph_3       dw 720*2 dup(?)
old_ax_3          dw ?
old_di_3          dw ?

index             dw ?
pixel             db ?

min_index_0       dw ?
min_index_1       dw ?
min_index_2       dw ?
min_index_3       dw ?

min_0             dw ?
min_1             dw ?
min_2             dw ?
min_3             dw ?

old_period_marker  dw ?
period_marker_base dw ?
old_period_pixel   db ?
period_pixel_base  db ?

left_history_buffer  db (X_COUNT * Y_COUNT) dup (?)
right_history_buffer db (X_COUNT * Y_COUNT) dup (?)

                  .code

                  extrn eight:byte

bank_number       db 0
x_sum             dd ?
y_sum             dd ?
count             dw ?
zone_count        dw ?

                  public _int3
_int3             proc near

                  int 3
                  ret

_int3             endp


                  public _get_period
_get_period       proc near

                  push di

                  mov di,CORR_SUMS_END

                  mov dx,0FFFFH              ;Init mins
```

```
                mov min_0,dx
                mov min_1,dx
                mov min_2,dx
                mov min_3,dx

                mov bx,0                        ;Init max
                mov cx,CORR_HISTORY_SIZE - 1

;*****************************************************************************
;               Find First Minimum Sum
;*****************************************************************************

rising_loop_0:
                get_a_sum
                cmp ax,bx
                jb not_new_max_0

                mov bx,ax
                loop rising_loop_0
                jmp find_best_min

not_new_max_0:
                sub ax,bx
                neg ax
                cmp ax,BANDGAP
                jae found_max_0

                loop rising_loop_0
                jmp find_best_min

found_max_0:
                loop falling_loop_0
                jmp find_best_min

falling_loop_0:
                get_a_sum
                cmp ax,dx
                jae not_new_min_0

                mov dx,ax
                mov min_index_0,cx

                loop falling_loop_0
                jmp find_best_min

not_new_min_0:
                sub ax,dx
                cmp ax,BANDGAP
                jae found_min_0

                loop falling_loop_0
                jmp find_best_min

found_min_0:
                mov min_0,dx
                mov dx,0FFFFH                   ;Init mins
                mov bx,0                        ;Init max
```

```
                        loop rising_loop_1
                        jmp find_best_min

;*****************************************************************************
;                       Find Second Minimum Sum
;*****************************************************************************

rising_loop_1:
                        get_a_sum
                        cmp ax,bx
                        jb not_new_max_1

                        mov bx,ax
                        loop rising_loop_1
                        jmp find_best_min

not_new_max_1:
                        sub ax,bx
                        neg ax
                        cmp ax,BANDGAP
                        jae found_max_1

                        loop rising_loop_1
                        jmp find_best_min

found_max_1:
                        loop falling_loop_1
                        jmp find_best_min

falling_loop_1:
                        get_a_sum
                        cmp ax,dx
                        jae not_new_min_1

                        mov dx,ax
                        mov min_index_1,cx

                        loop falling_loop_1
                        jmp find_best_min

not_new_min_1:
                        sub ax,dx
                        cmp ax,BANDGAP
                        jae find_best_min

                        loop falling_loop_1
                        jmp find_best_min

;*****************************************************************************
;                       Decide which minimum sum is best
;*****************************************************************************

find_best_min:
                        mov ax,min_0

                        cmp ax,dx
                        ja second_is_smallest
                        jz both_mins_equal
```

```
;*****************************************************************************
;               First one is smallest
;*****************************************************************************

first_is_best:
                cmp ax,MAX_GOOD_CORR
                jb first_is_ok
                jmp no_good_corr

first_is_ok:
                mov cx,min_index_0
                mov ax,CORR_HISTORY_SIZE
                sub ax,cx
                jmp end_fms

;*****************************************************************************
;               Second one is smallest
;*****************************************************************************

second_is_smallest:
                sub ax,dx                       ;First - Second
                cmp ax,DECISION_ZONE
                jle first_is_best

second_is_best:
                cmp dx,MAX_GOOD_CORR
                jb second_is_ok
                jmp no_good_corr

second_is_ok:
                mov cx,min_index_1
                mov ax,CORR_HISTORY_SIZE
                sub ax,cx
                jmp end_fms


both_mins_equal:
                cmp ax,0FFFFH
                jnz first_is_best

no_good_corr:
                mov ax,0

end_fms:
                pop di
                ret

_get_period     endp
```

```
;***********************************************************************
;                       compute_correlation(new_data)
;                                                  4
;                         SI ptr to sums
;                         DI ptr to squares
;                         BX for lookup
;                         BP ptr to wave history
;                         CX counter for history size
;
;***********************************************************************

                    public _compute_correlation
_compute_correlation proc near

                push bp
                mov bp,sp
                push si
                push di

                mov ax,[bp+4]              ;Get new data point

                lea si,_corr_sums_buffer    ;Setup pointers
                mov di,corr_squares_ptr
                mov bp,wave_history_ptr

                mov cx,CORR_HISTORY_SIZE   ;Initialize counter

                add bp,2                   ;Word increment history pointer
                cmp bp,WAVE_HISTORY_END
                jnz not_w_h_el
                lea bp,wave_history_buffer
not_w_h_el:
                mov [bp],ax                ;Store new data in history buffer
;***********************************************************************
;               Subtract current from historic data
;               Square the result
;               Subrtact old square from sum
;               Add new square to sum
;               Put new sum in buffer
;***********************************************************************

compute_corr_loop:
                mov bx,[bp]                ;Get historic data
                sub bx,ax                  ;Get difference between historic and new
                shl bx,1
                mov dx,squares[bx]         ;Find square by lookup

                mov bx,[di]                ;Get previous square
                sub [si],bx                ;Remove it from long sum
                sbb word ptr[si+2],0

                mov [di],dx                ;Store new square in squares buffer

                add [si],dx                ;Add new square to sum
                adc word ptr[si+2],0
```

```
;*****************************************************************************
;               Adjust pointers and handle wraparound
;*****************************************************************************

                add si,4                    ;Long increment sum pointer
                add di,2                    ;Word increment squares pointer
                add bp,2                    ;Word increment wave history pointer

                cmp di,CORR_SQUARES_END
                jnz not_c_s_e
                lea di,corr_squares_buffer
not_c_s_e:
                cmp bp,WAVE_HISTORY_END
                jnz not_w_h_e2
                lea bp,wave_history_buffer
not_w_h_e2:

                loop compute_corr_loop

;*****************************************************************************
;               Save pointers for later
;*****************************************************************************

                mov corr_squares_ptr,di
                mov wave_history_ptr,bp

                pop di
                pop si
                pop bp
                ret

_compute_correlation endp


;*****************************************************************************
;               display_corr_graphs(new_data, period)
;                                     4         6
;*****************************************************************************


                public _display_corr_graphs
_display_corr_graphs proc near

                push bp
                mov bp,sp
                push si
                push di

                mov es,herc_screen

                mov bx,index
                mov dl,pixel

;*****************************************************************************
;               Add current data to graph 0
;*****************************************************************************
```

```
                mov di,old_graph_0[bx]      ;Get di of old graph
                mov cx,old_graph_0[bx+2]    ;And count and direction

                cmp di,0
                jz do_points_0c

                cmp cx,0                    ;Is old line up,down or horizontal
                jl blank_up_0c
                jg blank_down_0c

                xor es:[di],dl              ;Horizontal(blank one dot)
                jmp do_points_0c

blank_down_0c:
                call vert_line_d
                jmp do_points_0c

blank_up_0c:
                neg cx
                call vert_line_u

do_points_0c:
                mov ax,[bp+4]
                cmp ax,64
                jbe in_range_0c
                mov ax,64

in_range_0c:
                mov cx,old_ax_0
                mov di,old_di_0
                sub cx,ax                   ;Get count and direction

                mov old_graph_0[bx],di      ;Stash DI
                mov old_graph_0[bx+2],cx    ;And CX in array
                mov old_ax_0,ax             ;Stash AX

                cmp cx,0                    ;Is line up,down or horizontal?
                jl line_up_0c
                jg line_down_0c

                xor es:[di],dl              ;Horizontal(one dot)
                jmp next_pixel_0c

line_down_0c:
                call vert_line_d
                jmp next_pixel_0c

line_up_0c:
                neg cx
                call vert_line_u

next_pixel_0c:
                mov old_di_0,di             ;Stash DI

                ror pixel,1                 ;Shift bit pattern
                jnc no_inc_c
                inc old_di_0
```

```
no_inc_c:
                add index,4
                cmp index,720*4
                jl graph_done_c

                mov index,0
                sub old_di_0,90

graph_done_c:

;*******************************************************************************
;               Display entire correlation on graph 1
;*******************************************************************************

                mov si,CORR_SUMS_END
                mov old_di_1,90*42
                mov old_ax_1,0

                mov corr_count,CORR_HISTORY_SIZE
                mov dl,10000000B
                mov bx,0

corr_graph_loop:
                mov di,old_graph_1[bx]      ;Get di of old graph
                mov cx,old_graph_1[bx+2]    ;And count and direction

                cmp di,0
                jz do_points_1c

                cmp cx,0                    ;Is old line up,down or horizontal
                jl blank_up_1c
                jg blank_down_1c

                xor es:[di],dl              ;Horizontal(blank one dot)
                jmp do_points_1c

blank_down_1c:
                call vert_line_d
                jmp do_points_1c

blank_up_1c:
                neg cx
                call vert_line_u

do_points_1c:
                mov ax,[si]

                mov cl,6
                shr ax,cl

                mov cx,[si+2]
                sub si,4

                cmp cx,0
                jz no_uw
                mov ax,64
                jmp in_range_1c
```

```
no_uw:
                cmp ax,64
                jbe in_range_1c
                mov ax,64

in_range_1c:
                mov cx,old_ax_1
                mov di,old_di_1
                sub cx,ax                       ;Get count and direction

                mov old_graph_1[bx],di          ;Stash DI
                mov old_graph_1[bx+2],cx        ;And CX in array
                mov old_ax_1,ax                 ;Stash AX
                mov old_di_1,di                 ;Stash DI

                cmp cx,0                        ;Is line up,down or horizontal?
                jl line_up_1c
                jg line_down_1c

                xor es:[di],dl                  ;Horizontal(one dot)
                jmp next_pixel_1c

line_down_1c:
                call vert_line_d
                jmp next_pixel_1c

line_up_1c:
                neg cx
                call vert_line_u

next_pixel_1c:
                mov old_di_1,di                 ;Stash DI

                ror dl,1                        ;Shift bit pattern
                jnc no_inc_1c
                inc old_di_1

no_inc_1c:
                add bx,4
                cmp bx,720*4
                jl dec_count_1c

                mov bx,0
                sub old_di_1,90

dec_count_1c:
                dec corr_count
                jz graph_done_1c
                jmp corr_graph_loop

graph_done_1c:

;*******************************************************************************
;               Now display detected period marker
;*******************************************************************************

                ;++++
                mov di,old_period_marker
                mov dl,old_period_pixel
                mov cx,10
                call vert_line_d

                mov ax,[bp+6]
                div eight

                mov cl,ah
                mov dl,10000000B
                ror dl,cl
                mov old_period_pixel,dl

                mov ah,0
                mov di,period_marker_base
                add di,ax
                mov old_period_marker,di

                mov cx,10
                call vert_line_d

                pop di
                pop si
                pop bp
                ret

_display_corr_graphs endp

;******************************************************************************
;******************************************************************************

                public _align_camera
_align_camera   proc near

                push di
                mov es,ivg_seg

ac_loop:
                mov bank_number,INITIAL_BANK
                set_bank
                mov di,UPPER_LEFT
                mov bx,Y_COUNT

ac_ly_loop:
                push di
                mov cx,X_COUNT
                mov al,0FFH

ac_lx_loop:
                mov byte ptr es:[di],0FFH
                add di,X_INC
                loop ac_lx_loop

                pop di

                add di,Y_INC
                jnc lbank_ok

                inc bank_number
                mov al,ACQUIRE_FLAG_ON
                add al,bank_number
                mov dx,LOW_CSR_PORT
                out dx,al
lbank_ok:
                dec bx
                jnz ac_ly_loop

;*****************************************************************************
;*****************************************************************************

                mov bank_number,INITIAL_BANK
                set_bank
                mov di,UPPER_RIGHT
                mov bx,Y_COUNT

ac_ry_loop:
                push di
                mov cx,X_COUNT
                mov al,0FFH

ac_rx_loop:
                mov byte ptr es:[di],0FFH
                sub di,X_INC
                loop ac_rx_loop

                pop di

                add di,Y_INC
                jnc rbank_ok

                inc bank_number
                mov al,ACQUIRE_FLAG_ON
                add al,bank_number
                mov dx,LOW_CSR_PORT
                out dx,al
rbank_ok:
                dec bx
                jnz ac_ry_loop

                mov ah,1
                int 16H
                jnz ac_end
                jmp ac_loop

ac_end:
                mov ah,0
                int 16H

                pop di
                ret

_align_camera   endp

;*****************************************************************************
;*                      init_history_buffer();
;*****************************************************************************
```

```
                public _init_history_buffers
_init_history_buffers proc near

                push ds
                push si
                push di

                push ds
                pop es
                lea di,left_history_buffer

                mov ds,ivg_seg
                mov si,UPPER_LEFT
                mov bx,X_COUNT

ih_lx_loop:
                mov bank_number,INITIAL_BANK
                set_bank
                mov cx,Y_COUNT
                push si

ih_ly_loop:
                mov al,[si]
                stosb
                inc_y
                loop ih_ly_loop

                pop si
                add si,X_INC
                dec bx
                jnz ih_lx_loop

;******************************************************************************
;******************************************************************************

                lea di,right_history_buffer

                mov si,UPPER_RIGHT
                mov bx,X_COUNT

ih_rx_loop:
                mov bank_number,INITIAL_BANK
                set_bank
                mov cx,Y_COUNT
                push si

ih_ry_loop:
                mov al,[si]
                stosb
                inc_y
                loop ih_ry_loop

                pop si
                sub si,X_INC
                dec bx
                jnz ih_rx_loop
                pop di
                pop si
                pop ds
                ret

 _init_history_buffers endp

;*****************************************************************************
;*
;*              int get_l_centroid(int *);
;*
;*   filtered_count = get_l_centroid(&filtered_x, &filtered_y);
;*                                             4             6
;*****************************************************************************

                public _get_l_centroid
_get_l_centroid proc near

                push bp
                mov bp,sp
                push si
                push di

                push ds

                push ds
                pop es

;*****************************************************************************
;               Find first qualifying pixel
;               And start filling history buffer
;*****************************************************************************

                lea di,left_history_buffer
                mov ds,ivg_seg
                mov si,UPPER_LEFT

                mov word ptr x_sum,0
                mov word ptr x_sum[2],0
                mov word ptr y_sum,0
                mov word ptr y_sum[2],0
                mov count,0

                mov bx,X_COUNT
fp_l_x_loop:
                mov bank_number,INITIAL_BANK
                set_bank
                mov cx,Y_COUNT
                push si

fp_l_y_loop:
                mov al,[si]
                mov ah,al
                sub ah,es:[di]
                stosb
```

```
                    cmp ah,LOW_THRESHOLD
                    jb not_qual_l_fp
                    cmp ah,HIGH_THRESHOLD
                    ja not_qual_l_fp

                    mov zone_count,ZONE_SIZE
                    jmp add_to_l_sums

  not_qual_l_fp:
  ;                  mov byte ptr[si],080H     ;+++ light scan rectangle dimly

                    inc_y
                    loop fp_l_y_loop

                    pop si
                    add si,X_INC
                    dec bx
                    jnz fp_l_x_loop

                    mov count,0
                    jmp compute_l_centroid

;*****************************************************************************
;                Get centroid of qualifying pixels
;*****************************************************************************

gc_l_x_loop:
                    mov bank_number,INITIAL_BANK
                    set_bank
                    mov cx,Y_COUNT
                    push si

gc_l_y_loop:
                    mov al,[si]
                    mov ah,al
                    sub ah,es:[di]
                    stosb

                    cmp ah,LOW_THRESHOLD
                    jb not_qual_l_gc
                    cmp ah,HIGH_THRESHOLD
                    ja not_qual_l_gc

add_to_l_sums:
                    mov byte ptr[si],0FFH

                    add word ptr x_sum,bx
                    adc word ptr x_sum[2],0

                    add word ptr y_sum,cx
                    adc word ptr y_sum[2],0

                    inc count

not_qual_l_gc:
                    inc_y
                    loop gc_l_y_loop
```

```
                pop si
                add si,X_INC

                dec bx
                jz compute_1_centroid

                dec zone_count
                jnz gc_1_x_loop

;*****************************************************************************
;               Finish filling history buffer
;*****************************************************************************

fh_1_x_loop:
                mov bank_number,INITIAL_BANK
                set_bank
                mov cx,Y_COUNT
                push si

fh_1_y_loop:
                mov al,[si]
                stosb
                inc_y
                loop fh_1_y_loop

                pop si
                add si,X_INC
                dec bx
                jnz fh_1_x_loop

;*****************************************************************************
;               Compute the centroid if possible
;               And filter it using moving window averaging
;*****************************************************************************

compute_1_centroid:
                pop ds
                shr count,1                 ;+++

                cmp count,MIN_COUNT
                jbe no_1_centroid

;*****************************************************************************
;               Compute and filter x centroid
;*****************************************************************************

                mov ax,word ptr x_sum
                mov dx,word ptr x_sum[2]
                div count

                mov si,1_x_filt_ptr

                mov bx,1_x_filt_sum
                sub bx,1_x_filt_buffer[si]
                mov 1_x_filt_buffer[si],ax

                add ax,bx
                mov l_x_filt_sum,ax
                mov cl,CENT_FILT_SHIFT
                shr ax,cl

                add si,2
                cmp si,CENT_FILT_SIZE*2
                jnz l_x_filt_ok
                mov si,0
l_x_filt_ok:
                mov l_x_filt_ptr,si

                mov di,[bp+4]
                mov [di],ax

;*****************************************************************************
;               Compute and filter y centroid
;*****************************************************************************

                mov ax,word ptr y_sum
                mov dx,word ptr y_sum[2]
                div count

                mov si,l_y_filt_ptr

                mov bx,l_y_filt_sum
                sub bx,l_y_filt_buffer[si]
                mov l_y_filt_buffer[si],ax

                add ax,bx
                mov l_y_filt_sum,ax
                mov cl,CENT_FILT_SHIFT
                shr ax,cl

                add si,2
                cmp si,CENT_FILT_SIZE*2
                jnz l_y_filt_ok
                mov si,0
l_y_filt_ok:
                mov l_y_filt_ptr,si

                mov di,[bp+6]
                mov [di],ax

;*****************************************************************************
;               Filter count using moving window averaging
;*****************************************************************************
no_l_centroid:
                mov si,l_count_filt_ptr

                mov ax,l_count_filt_sum
                sub ax,l_count_filt_buffer[si]
                mov bx,count
                mov l_count_filt_buffer[si],bx

                add ax,bx
                mov l_count_filt_sum,ax
                mov cl,COUNT_FILT_SHIFT
                shr ax,cl

                add si,2
                cmp si,COUNT_FILT_SIZE*2
                jnz l_count_filt_ok
                mov si,0
  l_count_filt_ok:
                mov l_count_filt_ptr,si


                pop di
                pop si
                pop bp
                ret

 _get_l_centroid      endp

;***************************************************************************
;*
;*          int get_r_centroid(int *);
;*
;*    filtered_count = get_r_centroid(&filtered_x, &filtered_y);
;*                                              4              6
;***************************************************************************

                public _get_r_centroid
_get_r_centroid proc near

                push bp
                mov bp,sp
                push si
                push di

                push ds

                push ds
                pop es

;***************************************************************************
;               Find first qualifying pixel
;               And start filling history buffer
;***************************************************************************

                lea di,right_history_buffer

                mov ds,ivg_seg
                mov si,UPPER_RIGHT

                mov word ptr x_sum,0
                mov word ptr x_sum[2],0
                mov word ptr y_sum,0
                mov word ptr y_sum[2],0
                mov count,0

                mov bx,X_COUNT

fp_r_x_loop:
```

```
                mov bank_number,INITIAL_BANK
                set_bank
                mov cx,Y_COUNT
                push si
fp_r_y_loop:
                mov al,[si]
                mov ah,al
                sub ah,es:[di]
                stosb

                cmp ah,LOW_THRESHOLD
                jb not_qual_r_fp
                cmp ah,HIGH_THRESHOLD
                ja not_qual_r_fp

                mov zone_count,ZONE_SIZE
                jmp add_to_r_sums
not_qual_r_fp:
;               mov byte ptr[si],080H      ;+++ light grid dimly

                inc_y
                loop fp_r_y_loop

                pop si
                sub si,X_INC
                dec bx
                jnz fp_r_x_loop

                mov count,0
                jmp compute_r_centroid

;**************************************************************************
;               Get centroid of qualifying pixels
;**************************************************************************
gc_r_x_loop:
                mov bank_number,INITIAL_BANK
                set_bank
                mov cx,Y_COUNT
                push si
gc_r_y_loop:
                mov al,[si]
                mov ah,al
                sub ah,es:[di]
                stosb

                cmp ah,LOW_THRESHOLD
                jb not_qual_r_gc
                cmp ah,HIGH_THRESHOLD
                ja not_qual_r_gc
add_to_r_sums:
                mov byte ptr[si],0FFH

                add word ptr x_sum,bx
```

```
                    adc word ptr x_sum[2],0

                    add word ptr y_sum,cx
                    adc word ptr y_sum[2],0

                    inc count

not_qual_r_gc:
                    inc_y
                    loop gc_r_y_loop

                    pop si
                    sub si,X_INC

                    dec bx
                    jz compute_r_centroid

                    dec zone_count
                    jnz gc_r_x_loop

;******************************************************************************
;                   Finish filling history buffer
;******************************************************************************

fh_r_x_loop:
                    mov bank_number,INITIAL_BANK
                    set_bank
                    mov cx,Y_COUNT
                    push si

fh_r_y_loop:
                    mov al,[si]
                    stosb
                    inc_y
                    loop fh_r_y_loop

                    pop si
                    sub si,X_INC
                    dec bx
                    jnz fh_r_x_loop

;******************************************************************************
;                   Compute the centroid if possible
;                   And filter it using moving window averaging
;******************************************************************************

compute_r_centroid:
                    pop ds
                    shr count,1                ;+++

                    cmp count,MIN_COUNT
                    jbe no_r_centroid

;******************************************************************************
;                   Compute and filter x centroid
;******************************************************************************
```

```
                mov ax,word ptr x_sum
                mov dx,word ptr x_sum[2]
                div count

                mov si,r_x_filt_ptr

                mov bx,r_x_filt_sum
                sub bx,r_x_filt_buffer[si]
                mov r_x_filt_buffer[si],ax

                add ax,bx
                mov r_x_filt_sum,ax
                mov cl,CENT_FILT_SHIFT
                shr ax,cl

                add si,2
                cmp si,CENT_FILT_SIZE*2
                jnz r_x_filt_ok
                mov si,0
r_x_filt_ok:
                mov r_x_filt_ptr,si

                mov di,[bp+4]
                mov [di],ax

;*****************************************************************************
;               Compute and filter y centroid
;*****************************************************************************

                mov ax,word ptr y_sum
                mov dx,word ptr y_sum[2]
                div count

                mov si,r_y_filt_ptr

                mov bx,r_y_filt_sum
                sub bx,r_y_filt_buffer[si]
                mov r_y_filt_buffer[si],ax

                add ax,bx
                mov r_y_filt_sum,ax
                mov cl,CENT_FILT_SHIFT
                shr ax,cl

                add si,2
                cmp si,CENT_FILT_SIZE*2
                jnz r_y_filt_ok
                mov si,0
r_y_filt_ok:
                mov r_y_filt_ptr,si

                mov di,[bp+6]
                mov [di],ax

;*****************************************************************************
;               Filter count using moving window averaging
;*****************************************************************************
```

```
  no_r_centroid:
                mov si,r_count_filt_ptr

                mov ax,r_count_filt_sum
                sub ax,r_count_filt_buffer[si]
                mov bx,count
                mov r_count_filt_buffer[si],bx

                add ax,bx
                mov r_count_filt_sum,ax
                mov cl,COUNT_FILT_SHIFT
                shr ax,cl

                add si,2
                cmp si,COUNT_FILT_SIZE*2
                jnz r_count_filt_ok
                mov si,0
 r_count_filt_ok:
                mov r_count_filt_ptr,si

                pop di
                pop si
                pop bp
                ret

 _get_r_centroid endp

;****************************************************************************
;****************************************************************************

                public _wait_for_odd_field
_wait_for_odd_field proc near

                mov dx,HIGH_CSR_PORT

wfef_loop:
                in al,dx
                test al,10000000B
                jnz wfef_loop                   ;loop while field is odd

wfof_loop:
                in al,dx
                test al,10000000B
                jz wfof_loop                    ;loop while field is even

                ret

_wait_for_odd_field endp

;****************************************************************************
;****************************************************************************

                public _init_sparkle_lut
_init_sparkle_lut proc near

                ;Set OLUT 0, Set ILUT 0, No Write Protect
```

```
                        mov al,00000000B
                        mov dx,HIGH_CSR_PORT
                        out dx,al

                        mov ax,0
        slut_loop:
                        mov dx,LUT_ADDRESS_PORT
                        out dx,al
                        mov dx,INPUT_LUT_DATA_PORT
                        out dx,al

                        xchg ah,al

                        mov dx,RED_LUT_DATA_PORT
                        out dx,al
                        mov dx,GREEN_LUT_DATA_PORT
                        out dx,al
                        mov dx,BLUE_LUT_DATA_PORT
                        out dx,al

                        xchg ah,al

                        inc al
                        jnz slut_loop


                        mov dx,LUT_ADDRESS_PORT
                        mov al,0FFH
                        out dx,al

                        mov dx,RED_LUT_DATA_PORT
                        out dx,al
                        mov dx,GREEN_LUT_DATA_PORT
                        out dx,al
                        mov dx,BLUE_LUT_DATA_PORT
                        out dx,al

                        dec al
                        mov dx,INPUT_LUT_DATA_PORT
                        out dx,al


                        mov al,ACQUIRE_FLAG_ON
                        mov dx,LOW_CSR_PORT
                        out dx,al

                        ret

_init_sparkle_lut endp



                  public _init_norm_lut
_init_norm_lut    proc near

                  ;Set OLUT 0, Set ILUT 0, No Write Protect
```

```
                mov al,00000000B
                mov dx,HIGH_CSR_PORT
                out dx,al

                mov al,0
nlut_loop:

                mov dx,LUT_ADDRESS_PORT
                out dx,al
                mov dx,RED_LUT_DATA_PORT
                out dx,al
                mov dx,GREEN_LUT_DATA_PORT
                out dx,al
                mov dx,BLUE_LUT_DATA_PORT
                out dx,al
                mov dx,INPUT_LUT_DATA_PORT
                out dx,al

                inc al
                jnz nlut_loop

                mov al,ACQUIRE_FLAG_ON
                mov dx,LOW_CSR_PORT
                out dx,al

                ret

_init_norm_lut endp


;*****************************************************************************
;*****************************************************************************

                public _init_corr_graphs
_init_corr_graphs proc near

                push di

                push ds
                pop es

                mov index,0
                mov pixel,10000000B
                mov ax,0

                mov old_di_0,90*21
                mov old_ax_0,0

                lea di,old_graph_0
                mov cx,720*2
                rep stosw

                mov old_di_1,90*42
                mov old_ax_1,0

                lea di,old_graph_1
                mov cx,720*2
                rep stosw
                lea di,_corr_sums_buffer
                mov cx,CORR_HISTORY_SIZE *2
                rep stosw

                lea di,corr_squares_buffer
                mov cx,SQUARES_BUFFER_SIZE
                rep stosw

                lea di,wave_history_buffer
                mov cx,CORR_HISTORY_SIZE
                rep stosw

                ;**************************

                mov old_period_pixel,10000000B

                mov old_period_marker,90*42
                mov period_marker_base,90*42

                pop di
                ret

_init_corr_graphs endp

;******************************************************************************
;******************************************************************************

                public _init_four_graphs
_init_four_graphs proc near

                push di

                push ds
                pop es

                mov index,0
                mov pixel,10000000B
                mov ax,0

                mov old_di_0,90*21
                mov old_ax_0,0

                lea di,old_graph_0
                mov cx,720*2
                rep stosw

                mov old_di_1,90*42
                mov old_ax_1,0

                lea di,old_graph_1
                mov cx,720*2
                rep stosw

                mov old_di_2,90*63
                mov old_ax_2,0
                lea di,old_graph_2
                mov cx,720*2
                rep stosw

                mov old_di_3,90*84
                mov old_ax_3,0

                lea di,old_graph_3
                mov cx,720*2
                rep stosw

                pop di
                ret

_init_four_graphs endp

;******************************************************************************
;               display_four_graphs(y0, y1, y2, y3)   where y's are from 0-64
;                                    4   6   8   10
;******************************************************************************


                public _display_four_graphs
_display_four_graphs proc near

                push bp
                mov bp,sp
                push di

                mov es,herc_screen

                mov bx,index
                mov dl,pixel

;******************************************************************************


blank_graph_0:
                mov di,old_graph_0[bx]      ;Get di of old graph
                mov cx,old_graph_0[bx+2]    ;And count and direction

                cmp di,0
                jz do_points_0

                cmp cx,0                    ;Is old line up,down or horizontal
                jl blank_up_0
                jg blank_down_0

                xor es:[di],dl              ;Horizontal(blank one dot)
                jmp do_points_0
blank_down_0:
                call vert_line_d
                jmp do_points_0
blank_up_0:
                neg cx
                call vert_line_u
```

```
do_points_0:
                mov ax,[bp+4]
                cmp ax,64
                jbe in_range_0
                mov ax,64
in_range_0:

                mov cx,old_ax_0
                mov di,old_di_0
                sub cx,ax                   ;Get count and direction

                mov old_graph_0[bx],di      ;Stash DI
                mov old_graph_0[bx+2],cx    ;And CX in array
                mov old_ax_0,ax             ;Stash AX

                cmp cx,0                    ;Is line up,down or horizontal?
                jl line_up_0
                jg line_down_0

                xor es:[di],dl              ;Horizontal(one dot)
                jmp blank_graph_1

line_down_0:
                call vert_line_d
                jmp blank_graph_1

line_up_0:
                neg cx
                call vert_line_u

;*****************************************************************************

blank_graph_1:
                mov old_di_0,di             ;Stash DI

                mov di,old_graph_1[bx]      ;Get di of old graph
                mov cx,old_graph_1[bx+2]    ;And count and direction

                cmp di,0
                jz do_points_1

                cmp cx,0                    ;Is old line up,down or horizontal
                jl blank_up_1
                jg blank_down_1

                xor es:[di],dl              ;Horizontal(blank one dot)
                jmp do_points_1

blank_down_1:
                call vert_line_d
                jmp do_points_1

blank_up_1:
                neg cx
                call vert_line_u

do_points_1:
```

```
                mov ax,[bp+6]
                cmp ax,64
                jbe in_range_1
                mov ax,64
in_range_1:

                mov cx,old_ax_1
                mov di,old_di_1
                sub cx,ax                          ;Get count and direction

                mov old_graph_1[bx],di             ;Stash DI
                mov old_graph_1[bx+2],cx           ;And CX in array
                mov old_ax_1,ax                    ;Stash AX
                mov old_di_1,di                    ;Stash DI

                cmp cx,0                           ;Is line up,down or horizontal?
                jl line_up_1
                jg line_down_1

                xor es:[di],dl                     ;Horizontal(one dot)
                jmp blank_graph_2

line_down_1:
                call vert_line_d
                jmp blank_graph_2

line_up_1:
                neg cx
                call vert_line_u
;*****************************************************************************************

blank_graph_2:
                mov old_di_1,di                    ;Stash DI

                mov di,old_graph_2[bx]             ;Get di of old graph
                mov cx,old_graph_2[bx+2]           ;And count and direction

                cmp di,0
                jz do_points_2

                cmp cx,0                           ;Is old line up,down or horizontal
                jl blank_up_2
                jg blank_down_2

                xor es:[di],dl                     ;Horizontal(blank one dot)
                jmp do_points_2

blank_down_2:
                call vert_line_d
                jmp do_points_2

blank_up_2:
                neg cx
                call vert_line_u

do_points_2:
                mov ax,[bp+8]
                cmp ax,64
                jbe in_range_2
                mov ax,64
in_range_2:

                mov cx,old_ax_2
                mov di,old_di_2
                sub cx,ax                     ;Get count and direction

                mov old_graph_2[bx],di        ;Stash DI
                mov old_graph_2[bx+2],cx      ;And CX in array
                mov old_ax_2,ax               ;Stash AX
                mov old_di_2,di               ;Stash DI

                cmp cx,0                      ;Is line up,down or horizontal?
                jl line_up_2
                jg line_down_2

                xor es:[di],dl                ;Horizontal(one dot)
                jmp blank_graph_3

line_down_2:
                call vert_line_d
                jmp blank_graph_3

line_up_2:
                neg cx
                call vert_line_u

;*****************************************************************************

blank_graph_3:
                mov old_di_2,di               ;Stash DI

                mov di,old_graph_3[bx]        ;Get di of old graph
                mov cx,old_graph_3[bx+2]      ;And count and direction

                cmp di,0
                jz do_points_3

                cmp cx,0                      ;Is old line up,down or horizontal
                jl blank_up_3
                jg blank_down_3

                xor es:[di],dl                ;Horizontal(blank one dot)
                jmp do_points_3

blank_down_3:
                call vert_line_d
                jmp do_points_3

blank_up_3:
                neg cx
                call vert_line_u

do_points_3:
                mov ax,[bp+10]
                cmp ax,64
```

```
                    jbe in_range_3
                    mov ax,64
    in_range_3:

                    mov cx,old_ax_3
                    mov di,old_di_3
                    sub cx,ax                   ;Get count and direction

                    mov old_graph_3[bx],di      ;Stash DI
                    mov old_graph_3[bx+2],cx    ;And CX in array
                    mov old_ax_3,ax             ;Stash AX
                    mov old_di_3,di             ;Stash DI

                    cmp cx,0                    ;Is line up,down or horizontal?
                    jl line_up_3
                    jg line_down_3

                    xor es:[di],dl              ;Horizontal(one dot)
                    jmp next_pixel

    line_down_3:
                    call vert_line_d
                    jmp next_pixel

    line_up_3:
                    neg cx
                    call vert_line_u

    next_pixel:
                    mov old_di_3,di             ;Stash DI

                    ror pixel,1                 ;Shift bit pattern
                    jnc no_inc
                    inc old_di_0
                    inc old_di_1
                    inc old_di_2
                    inc old_di_3

    no_inc:
                    add index,4
                    cmp index,720*4
                    jl graph_done

                    mov index,0
                    sub old_di_0,90
                    sub old_di_1,90
                    sub old_di_2,90
                    sub old_di_3,90

    graph_done:
                    pop di
                    pop bp
                    ret

    _display_four_graphs endp
```

```
;*******************************************************************************
;*        Draw a vertical line DOWN from dotpos di for length cx
;*                       Using pattern in dl
;*******************************************************************************

vert_line_d     proc near

                cmp di,6000H
                jge line3

                cmp di,4000H
                jge line2

                cmp di,2000H
                jge line1

line0:
                xor es:[di], dl
                add di, 2000H
                loop line1
                ret
line1:
                xor es:[di], dl
                add di, 2000H
                loop line2
                ret
line2:
                xor es:[di], dl
                add di, 2000H
                loop line3
                ret
line3:
                xor es:[di], dl
                sub di, 5fa6H
                loop line0
                ret

vert_line_d     endp

;*******************************************************************************
;*        Draw a vertical line UP from dotpos di for length cx
;*                       Using pattern in dl
;*******************************************************************************

vert_line_u     proc near

                cmp di,2000H
                jl line0_u

                cmp di,4000H
                jl line1_u

                cmp di,6000H
                jl line2_u

line3_u:
                xor es:[di], dl
```

```
                    sub di, 2000H
                    loop line2_u
                    ret
    line2_u:
                    xor es:[di], dl
                    sub di, 2000H
                    loop line1_u
                    ret
    line1_u:
                    xor es:[di], dl
                    sub di, 2000H
                    loop line0_u
                    ret
    line0_u:
                    xor es:[di], dl
                    add di, 5fa6H
                    loop line3_u
                    ret

    vert_line_u     endp

                    end


                    public x_get_period
    x_get_period     proc near

                    push di

                    mov di,CORR_SUMS_END

                    mov dx,0FFFFH               ;Init mins
                    mov min_0,dx
                    mov min_1,dx
                    mov min_2,dx
                    mov min_3,dx

                    mov bx,0                    ;Init max
                    mov cx,CORR_HISTORY_SIZE - 1
;*****************************************************************************
;                   Find First Minimum Sum
;*****************************************************************************
rising_loop_0:
                    get_a_sum
                    cmp ax,bx
                    jb not_new_max_0

                    mov bx,ax
                    loop rising_loop_0
                    jmp find_best_min

not_new_max_0:
                    sub ax,bx
                    neg ax
                    cmp ax,BANDGAP
                    jae found_max_0

                    loop rising_loop_0
                    jmp find_best_min
     found_max_0:
                    loop falling_loop_0
                    jmp find_best_min
     falling_loop_0:
                    get_a_sum
                    cmp ax,dx
                    jae not_new_min_0

                    mov dx,ax
                    mov min_index_0,cx

                    loop falling_loop_0
                    jmp find_best_min
     not_new_min_0:
                    sub ax,dx
                    cmp ax,BANDGAP
                    jae found_min_0

                    loop falling_loop_0
                    jmp find_best_min
     found_min_0:
                    mov min_0,dx
                    mov dx,0FFFFH                 ;Init mins
                    mov bx,0                      ;Init max

                    loop rising_loop_1
                    jmp find_best_min
;*******************************************************************************
;                Find Second Minimum Sum
;*******************************************************************************
rising_loop_1:
                    get_a_sum
                    cmp ax,bx
                    jb not_new_max_1

                    mov bx,ax
                    loop rising_loop_1
                    jmp find_best_min
not_new_max_1:
                    sub ax,bx
                    neg ax
                    cmp ax,BANDGAP
                    jae found_max_1

                    loop rising_loop_1
                    jmp find_best_min
```

```
found_max_1:
                loop falling_loop_1
                jmp find_best_min

falling_loop_1:
                get_a_sum
                cmp ax,dx
                jae not_new_min_1

                mov dx,ax
                mov min_index_1,cx

                loop falling_loop_1
                jmp find_best_min

not_new_min_1:
                sub ax,dx
                cmp ax,BANDGAP
                jae found_min_1

                loop falling_loop_1
                jmp find_best_min

found_min_1:
                mov min_1,dx
                mov dx,0FFFFH               ;Init mins
                mov bx,0                    ;Init max

                loop rising_loop_2
                jmp find_best_min

;*******************************************************************************
;               Find Third Minimum Sum
;*******************************************************************************

rising_loop_2:
                get_a_sum
                cmp ax,bx
                jb not_new_max_2

                mov bx,ax
                loop rising_loop_2
                jmp find_best_min

not_new_max_2:
                sub ax,bx
                neg ax
                cmp ax,BANDGAP
                jae found_max_2

                loop rising_loop_2
                jmp find_best_min

found_max_2:
                loop falling_loop_2
                jmp find_best_min

falling_loop_2:
```

```
                get_a_sum
                cmp ax,dx
                jae not_new_min_2

                mov dx,ax
                mov min_index_2,cx

                loop falling_loop_2
                jmp find_best_min
not_new_min_2:
                sub ax,dx
                cmp ax,BANDGAP
                jae found_min_2

                loop falling_loop_2
                jmp find_best_min
found_min_2:
                mov min_2,dx
                mov dx,0FFFFH               ;Init mins
                mov bx,0                    ;Init max

                loop rising_loop_3
                jmp find_best_min
;*****************************************************************************
;               Find Fourth Minimum Sum
;*****************************************************************************
rising_loop_3:
                get_a_sum
                cmp ax,bx
                jb not_new_max_3

                mov bx,ax
                loop rising_loop_3
                jmp find_best_min
not_new_max_3:
                sub ax,bx
                neg ax
                cmp ax,BANDGAP
                jae found_max_3

                loop rising_loop_3
                jmp find_best_min
found_max_3:
                loop falling_loop_3
                jmp find_best_min
falling_loop_3:
                get_a_sum
                cmp ax,dx
                jae not_new_min_3

                mov dx,ax
                jle first_is_best
second_is_best:
                mov cx,min_index_1
                cmp cx,MAX_GOOD_CORR
                jb second_is_ok
                jmp no_good_corr

second_is_ok:
                mov ax,CORR_HISTORY_SIZE
                sub ax,cx
                jmp end_fms


second_not_smallest:
                cmp cx,dx
                ja third_not_smallest
                jz fourth_not_smallest

;*****************************************************************************
;               Third one is smallest
;*****************************************************************************

                sub ax,cx                       ;First - Third
                cmp ax,DECISION_ZONE
                jle first_is_best

                sub bx,cx                       ;Second - Third
                cmp bx,DECISION_ZONE
                jle second_is_best
third_is_best:
                mov cx,min_index_2
                cmp cx,MAX_GOOD_CORR
                jb third_is_ok
                jmp no_good_corr

third_is_ok:
                mov ax,CORR_HISTORY_SIZE
                sub ax,cx
                jmp end_fms

third_not_smallest:
                cmp cx,dx
                ja fourth_is_smallest
                jz fourth_not_smallest

;*****************************************************************************
;               Fourth One Is Smallest
;*****************************************************************************

fourth_is_smallest:
                sub ax,dx                       ;First - Fourth
                cmp ax,DECISION_ZONE
                jle first_is_best

                sub bx,dx                       ;Second - Fourth
                cmp bx,DECISION_ZONE
                mov min_index_3,cx

                loop falling_loop_3
                jmp find_best_min

not_new_min_3:
                sub ax,dx
                cmp ax,BANDGAP
                jae find_best_min

                loop falling_loop_3

;***************************************************************************
;               Decide which one is best
;***************************************************************************

find_best_min:
                mov ax,min_0
                mov bx,min_1
                mov cx,min_2

                cmp ax,bx
                jae first_not_smallest

                cmp ax,cx
                jae first_not_smallest

                cmp ax,dx
                jae first_not_smallest

;***************************************************************************
;               First one is smallest, and always best
;***************************************************************************

first_is_best:
                mov cx,min_index_0
                cmp cx,MAX_GOOD_CORR
                jb first_is_ok
                jmp no_good_corr

first_is_ok:
                mov ax,CORR_HISTORY_SIZE
                sub ax,cx
                jmp end_fms

first_not_smallest:
;               cmp bx,cx
;               jae second_not_smallest
;+++ ignore third & fourth correlations
;               cmp bx,dx
;               jae second_not_smallest

;***************************************************************************
;               Second one is smallest
;***************************************************************************

                sub ax,bx               ;First - Second
                cmp ax,DECISION_ZONE
```

```
                    jle second_is_best

                    sub cx,dx              ;Third - Fourth
                    cmp cx,DECISION_ZONE
                    jle third_is_best
 fourth_is_best:
                    mov cx,min_index_3
                    cmp cx,MAX_GOOD_CORR
                    jb fourth_is_ok
                    jmp no_good_corr

 fourth_is_ok:
                    mov ax,CORR_HISTORY_SIZE
                    sub ax,cx
                    jmp end_fms


 fourth_not_smallest:
                    cmp ax,0FFFFH
                    jz no_good_corr
                    jmp first_is_best

 no_good_corr:
                    mov ax,0

 end_fms:
                    pop di
                    ret

 x_get_period       endp
```

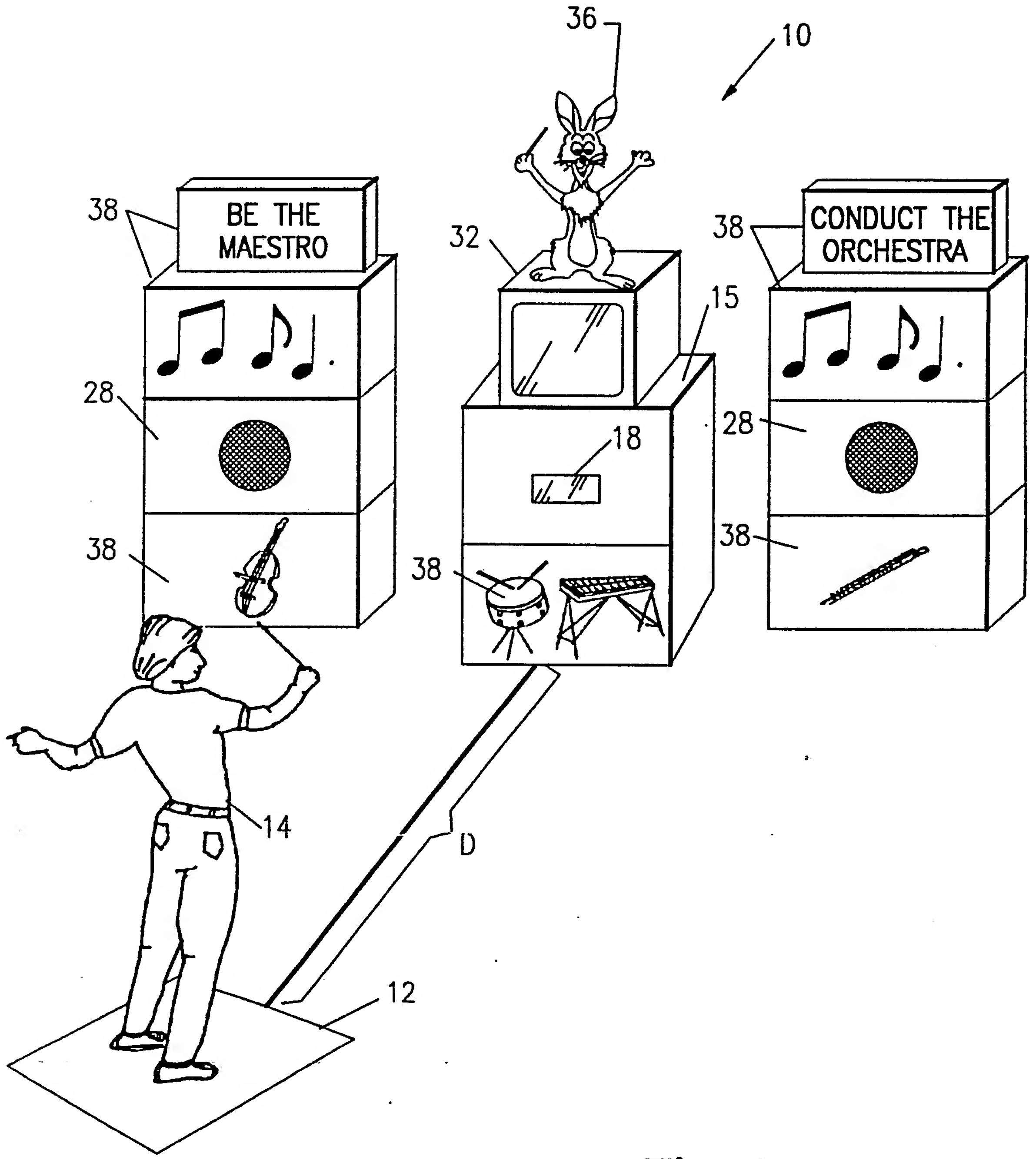

36
10
38
BE THE MAESTRO
32
38
CONDUCT THE ORCHESTRA
15
28
18
28
38
38
38
14
D
12

Fig.1

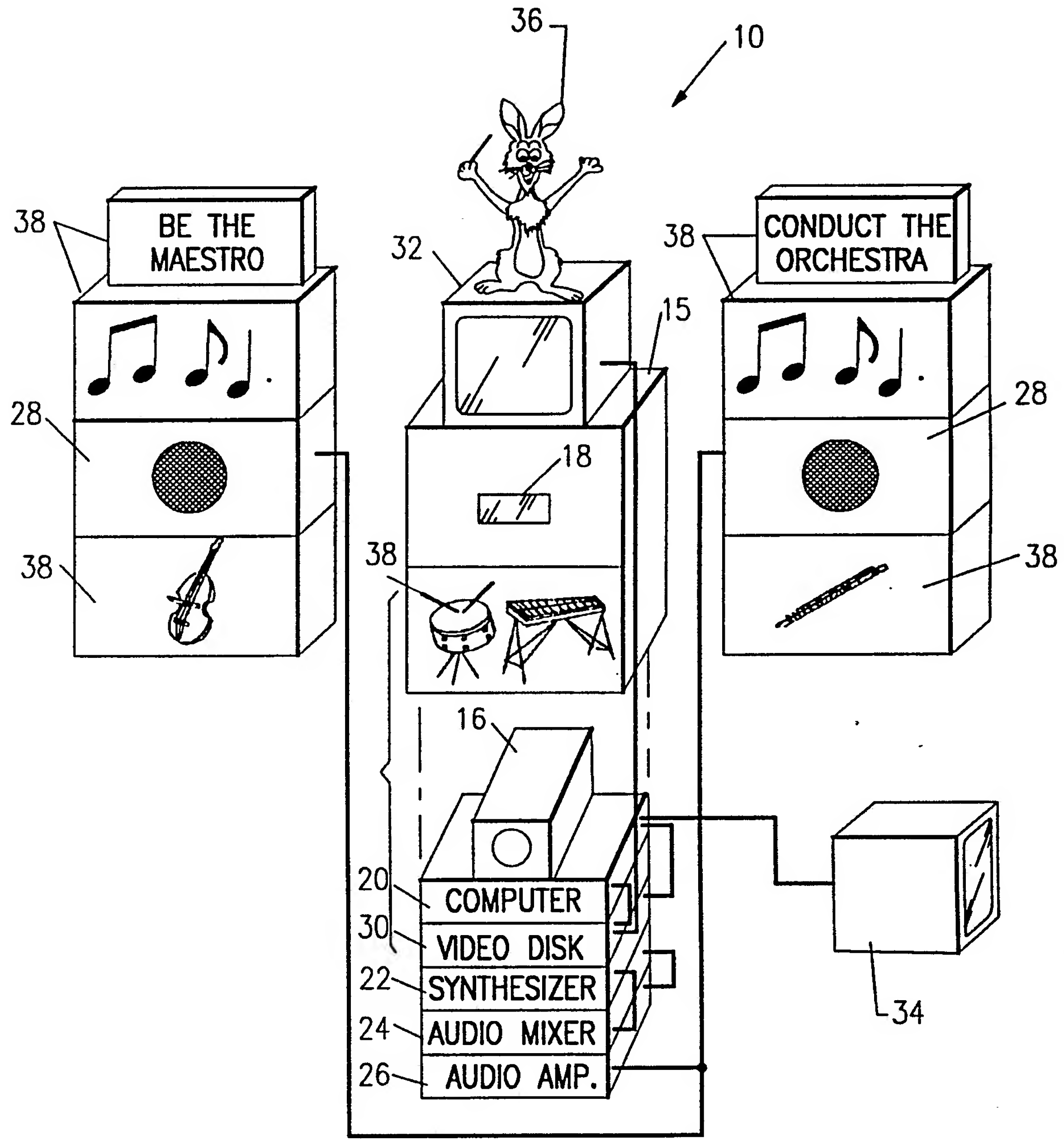

36
10
38
BE THE MAESTRO
32
CONDUCT THE ORCHESTRA
15
28
18
38
16
20
COMPUTER
30
VIDEO DISK
22
SYNTHESIZER
24
AUDIO MIXER
26
AUDIO AMP.
34

Fig.2

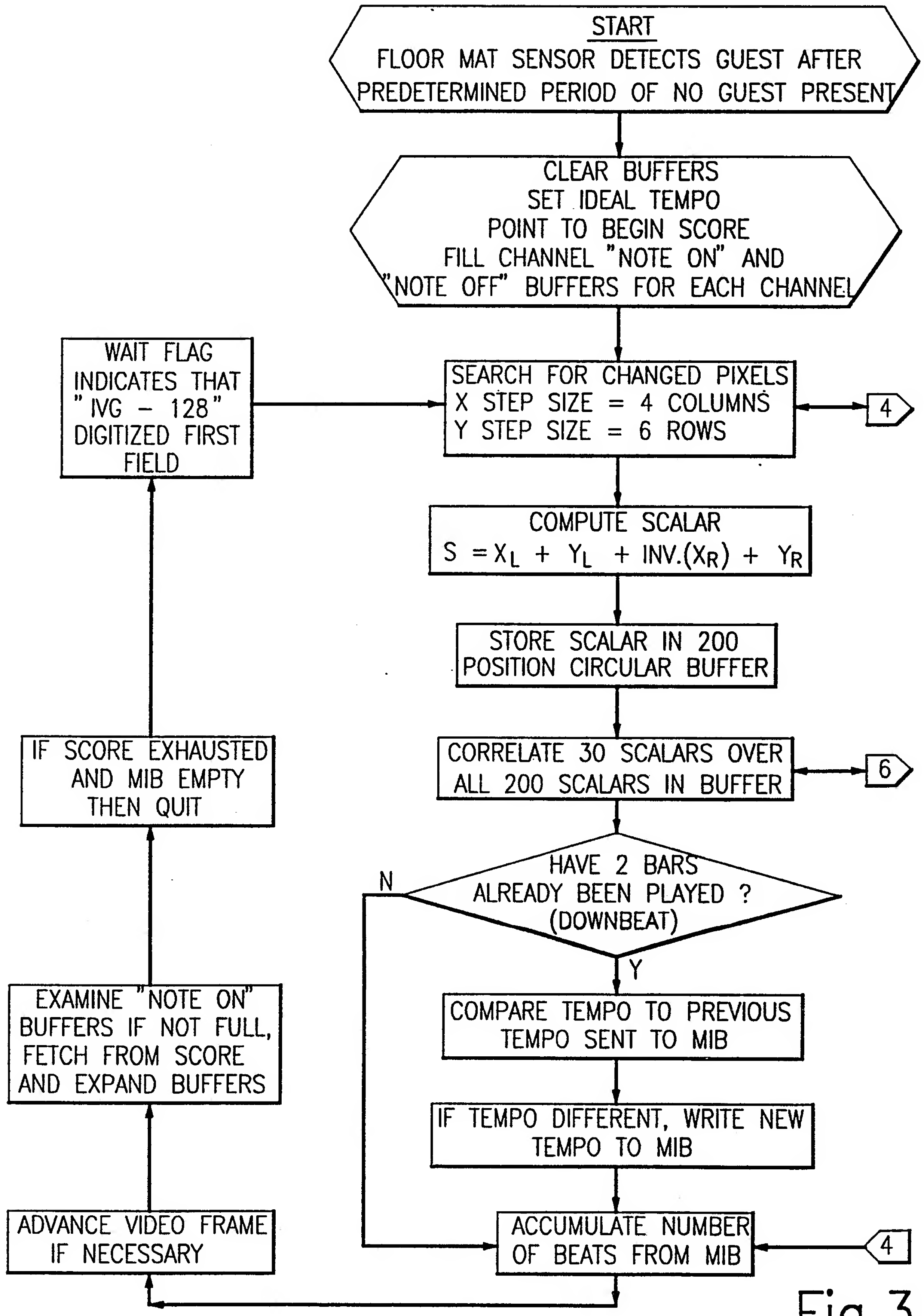

START
FLOOR MAT SENSOR DETECTS GUEST AFTER PREDETERMINED PERIOD OF NO GUEST PRESENT
CLEAR BUFFERS
SET IDEAL TEMPO
POINT TO BEGIN SCORE
FILL CHANNEL "NOTE ON" AND "NOTE OFF" BUFFERS FOR EACH CHANNEL
SEARCH FOR CHANGED PIXELS
X STEP SIZE = 4 COLUMNS
Y STEP SIZE = 6 ROWS
4
COMPUTE SCALAR
S = X_L + Y_L + INV.(X_R) + Y_R
STORE SCALAR IN 200 POSITION CIRCULAR BUFFER
CORRELATE 30 SCALARS OVER ALL 200 SCALARS IN BUFFER
6
HAVE 2 BARS ALREADY BEEN PLAYED ? (DOWNBEAT)
N
Y
COMPARE TEMPO TO PREVIOUS TEMPO SENT TO MIB
IF TEMPO DIFFERENT, WRITE NEW TEMPO TO MIB
ACCUMULATE NUMBER OF BEATS FROM MIB
4
ADVANCE VIDEO FRAME IF NECESSARY
EXAMINE "NOTE ON" BUFFERS IF NOT FULL, FETCH FROM SCORE AND EXPAND BUFFERS
IF SCORE EXHAUSTED AND MIB EMPTY THEN QUIT
WAIT FLAG INDICATES THAT "IVG - 128" DIGITIZED FIRST FIELD

Fig.3

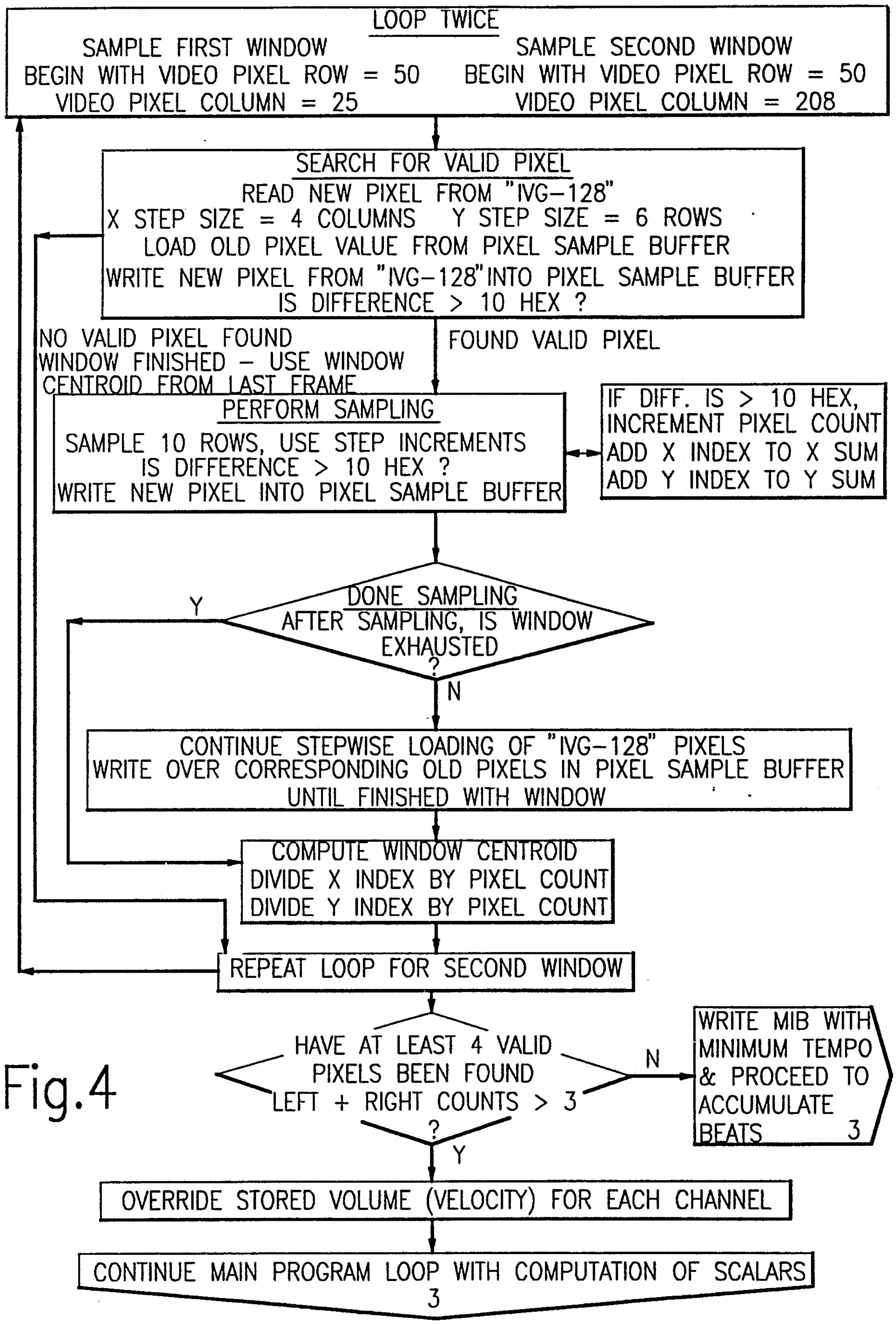

LOOP TWICE
SAMPLE FIRST WINDOW
BEGIN WITH VIDEO PIXEL ROW = 50
VIDEO PIXEL COLUMN = 25
SAMPLE SECOND WINDOW
BEGIN WITH VIDEO PIXEL ROW = 50
VIDEO PIXEL COLUMN = 208
SEARCH FOR VALID PIXEL
READ NEW PIXEL FROM "IVG-128"
X STEP SIZE = 4 COLUMNS Y STEP SIZE = 6 ROWS
LOAD OLD PIXEL VALUE FROM PIXEL SAMPLE BUFFER
WRITE NEW PIXEL FROM "IVG-128" INTO PIXEL SAMPLE BUFFER
IS DIFFERENCE > 10 HEX ?
NO VALID PIXEL FOUND
WINDOW FINISHED - USE WINDOW
CENTROID FROM LAST FRAME
FOUND VALID PIXEL
PERFORM SAMPLING
SAMPLE 10 ROWS, USE STEP INCREMENTS
IS DIFFERENCE > 10 HEX ?
WRITE NEW PIXEL INTO PIXEL SAMPLE BUFFER
IF DIFF. IS > 10 HEX,
INCREMENT PIXEL COUNT
ADD X INDEX TO X SUM
ADD Y INDEX TO Y SUM
DONE SAMPLING
AFTER SAMPLING, IS WINDOW
EXHAUSTED
?
Y
N
CONTINUE STEPWISE LOADING OF "IVG-128" PIXELS
WRITE OVER CORRESPONDING OLD PIXELS IN PIXEL SAMPLE BUFFER
UNTIL FINISHED WITH WINDOW
COMPUTE WINDOW CENTROID
DIVIDE X INDEX BY PIXEL COUNT
DIVIDE Y INDEX BY PIXEL COUNT
REPEAT LOOP FOR SECOND WINDOW
HAVE AT LEAST 4 VALID
PIXELS BEEN FOUND
LEFT + RIGHT COUNTS > 3
?
N
WRITE MIB WITH
MINIMUM TEMPO
& PROCEED TO
ACCUMULATE
BEATS
3
Y
OVERRIDE STORED VOLUME (VELOCITY) FOR EACH CHANNEL
CONTINUE MAIN PROGRAM LOOP WITH COMPUTATION OF SCALARS
3

Fig.4

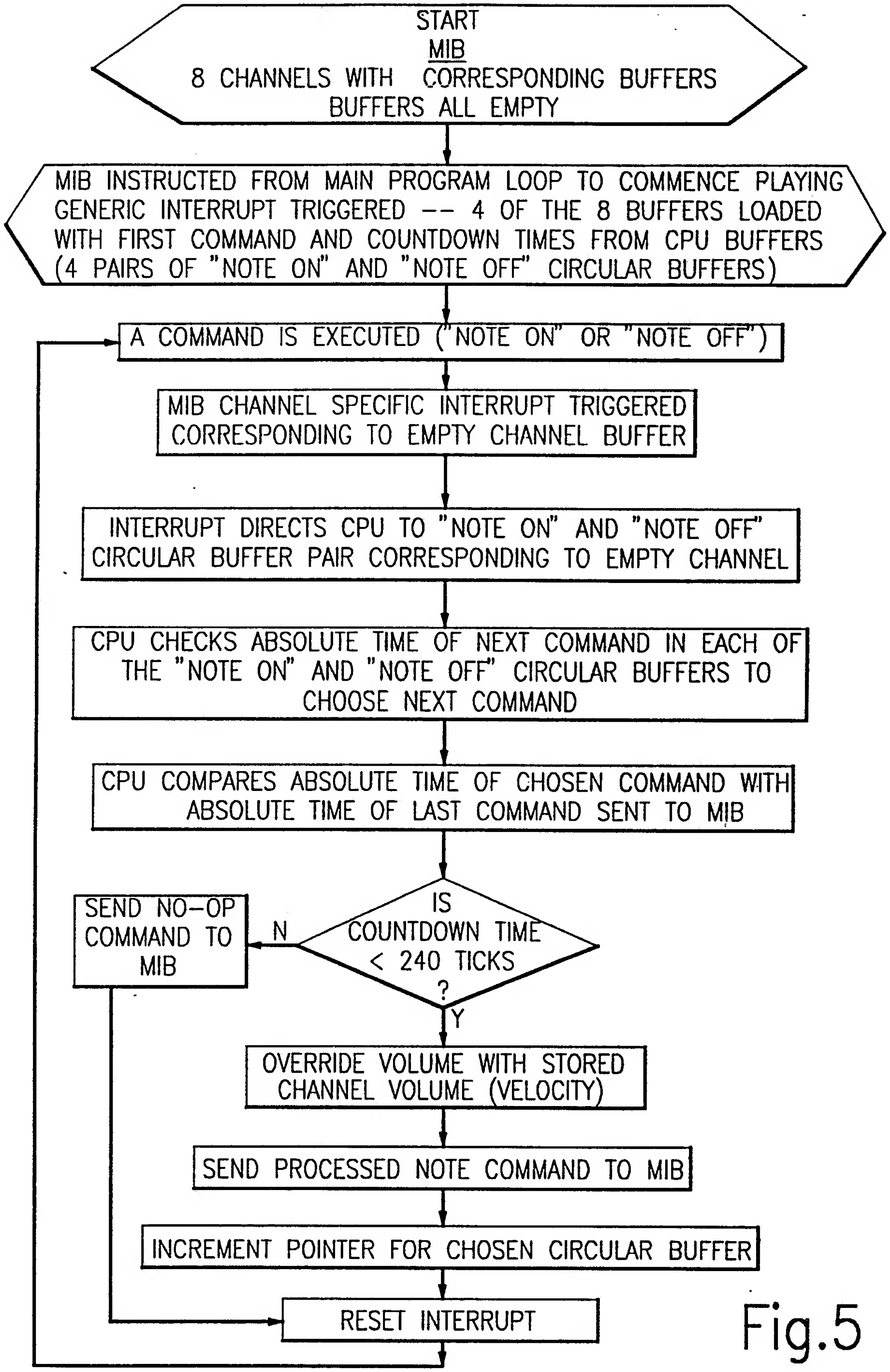

START
MIB
8 CHANNELS WITH CORRESPONDING BUFFERS
BUFFERS ALL EMPTY
MIB INSTRUCTED FROM MAIN PROGRAM LOOP TO COMMENCE PLAYING
GENERIC INTERRUPT TRIGGERED -- 4 OF THE 8 BUFFERS LOADED
WITH FIRST COMMAND AND COUNTDOWN TIMES FROM CPU BUFFERS
(4 PAIRS OF "NOTE ON" AND "NOTE OFF" CIRCULAR BUFFERS)
A COMMAND IS EXECUTED ("NOTE ON" OR "NOTE OFF")
MIB CHANNEL SPECIFIC INTERRUPT TRIGGERED CORRESPONDING TO EMPTY CHANNEL BUFFER
INTERRUPT DIRECTS CPU TO "NOTE ON" AND "NOTE OFF" CIRCULAR BUFFER PAIR CORRESPONDING TO EMPTY CHANNEL
CPU CHECKS ABSOLUTE TIME OF NEXT COMMAND IN EACH OF THE "NOTE ON" AND "NOTE OFF" CIRCULAR BUFFERS TO CHOOSE NEXT COMMAND
CPU COMPARES ABSOLUTE TIME OF CHOSEN COMMAND WITH ABSOLUTE TIME OF LAST COMMAND SENT TO MIB
IS COUNTDOWN TIME < 240 TICKS ?
N
SEND NO-OP COMMAND TO MIB
Y
OVERRIDE VOLUME WITH STORED CHANNEL VOLUME (VELOCITY)
SEND PROCESSED NOTE COMMAND TO MIB
INCREMENT POINTER FOR CHOSEN CIRCULAR BUFFER
RESET INTERRUPT

Fig.5

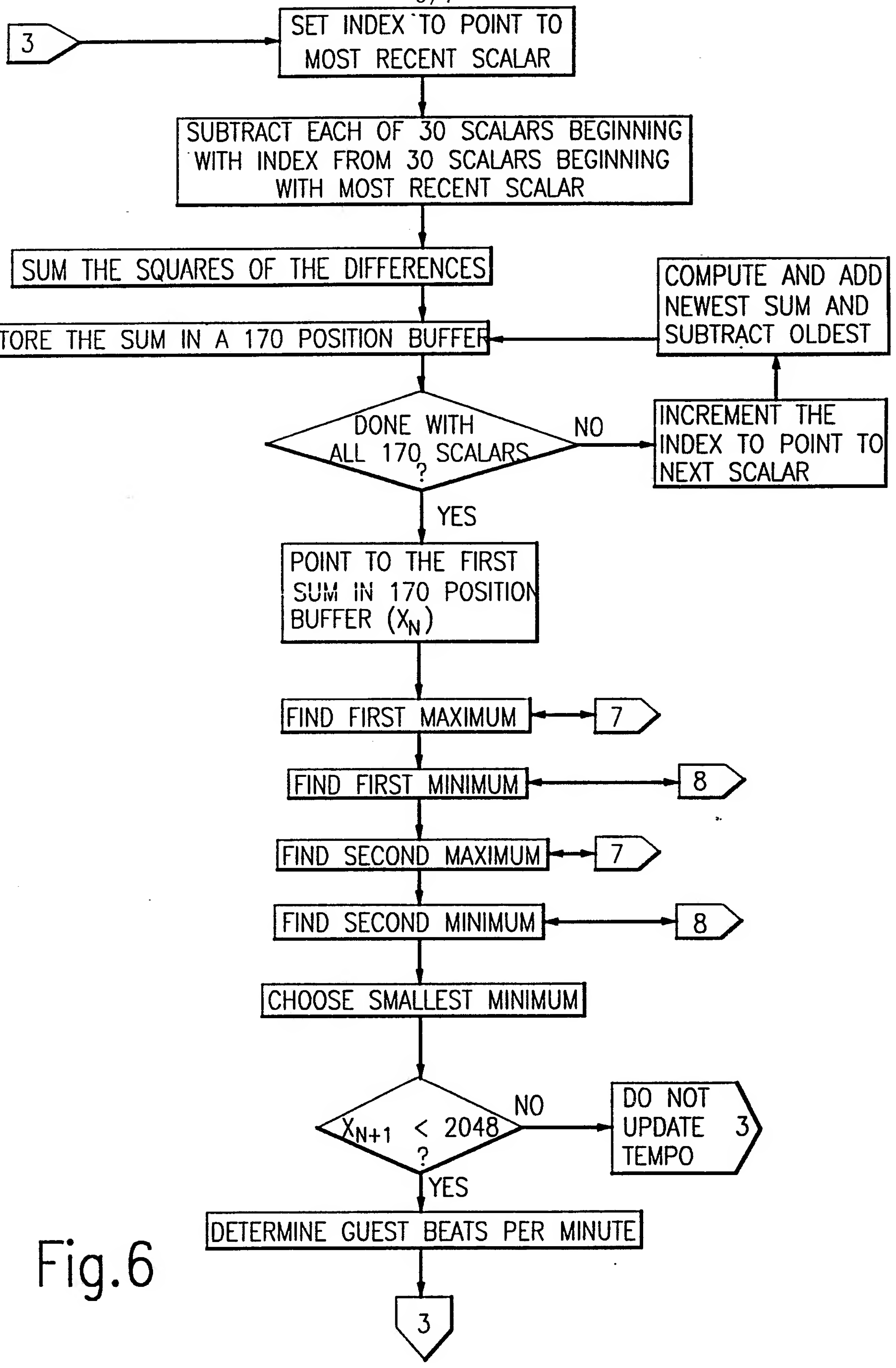

3
SET INDEX TO POINT TO MOST RECENT SCALAR
SUBTRACT EACH OF 30 SCALARS BEGINNING WITH INDEX FROM 30 SCALARS BEGINNING WITH MOST RECENT SCALAR
SUM THE SQUARES OF THE DIFFERENCES
STORE THE SUM IN A 170 POSITION BUFFER
COMPUTE AND ADD NEWEST SUM AND SUBTRACT OLDEST
DONE WITH ALL 170 SCALARS ?
NO
INCREMENT THE INDEX TO POINT TO NEXT SCALAR
YES
POINT TO THE FIRST SUM IN 170 POSITION BUFFER (X_N)
FIND FIRST MAXIMUM
7
FIND FIRST MINIMUM
8
FIND SECOND MAXIMUM
7
FIND SECOND MINIMUM
8
CHOOSE SMALLEST MINIMUM
$X_{N+1} < 2048$?
NO
DO NOT UPDATE TEMPO 3
YES
DETERMINE GUEST BEATS PER MINUTE
3

Fig.6

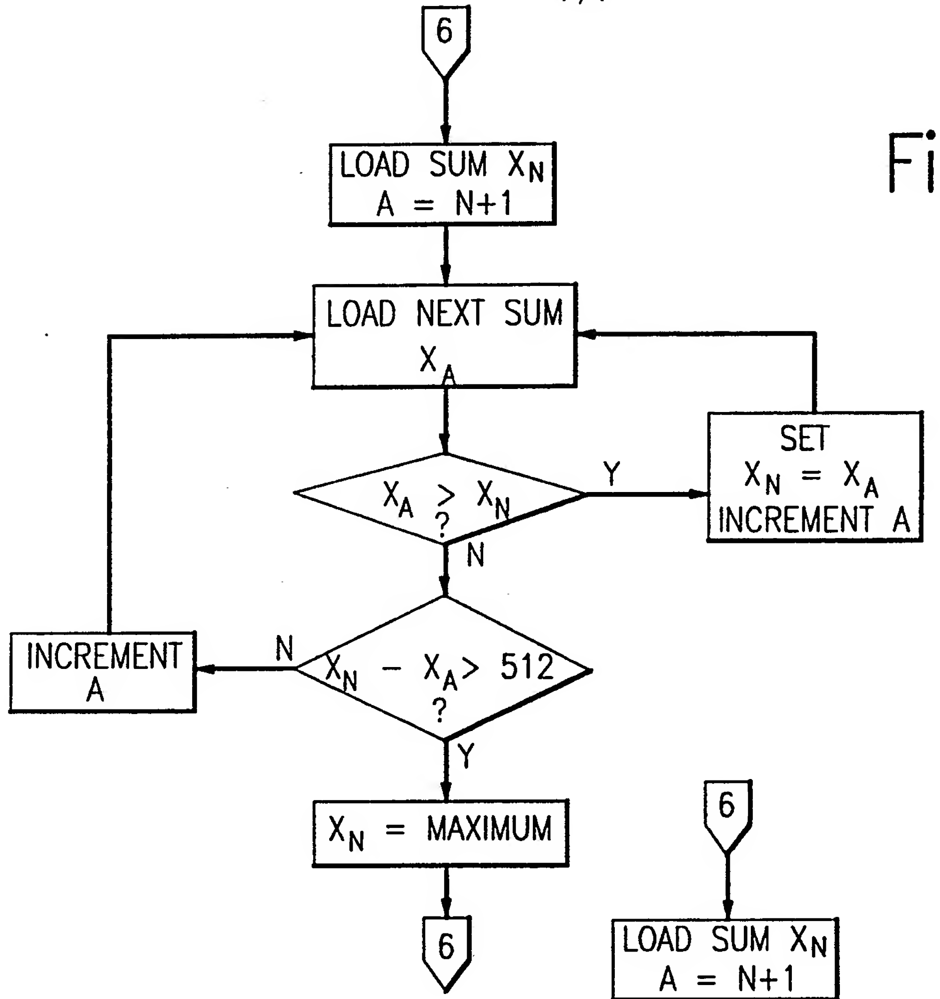

6
LOAD SUM X_N
A = N+1
LOAD NEXT SUM X_A
$X_A > X_N$?
Y
SET $X_N = X_A$ INCREMENT A
N
$X_N - X_A > 512$?
N
INCREMENT A
Y
X_N = MAXIMUM
6

Fig.7

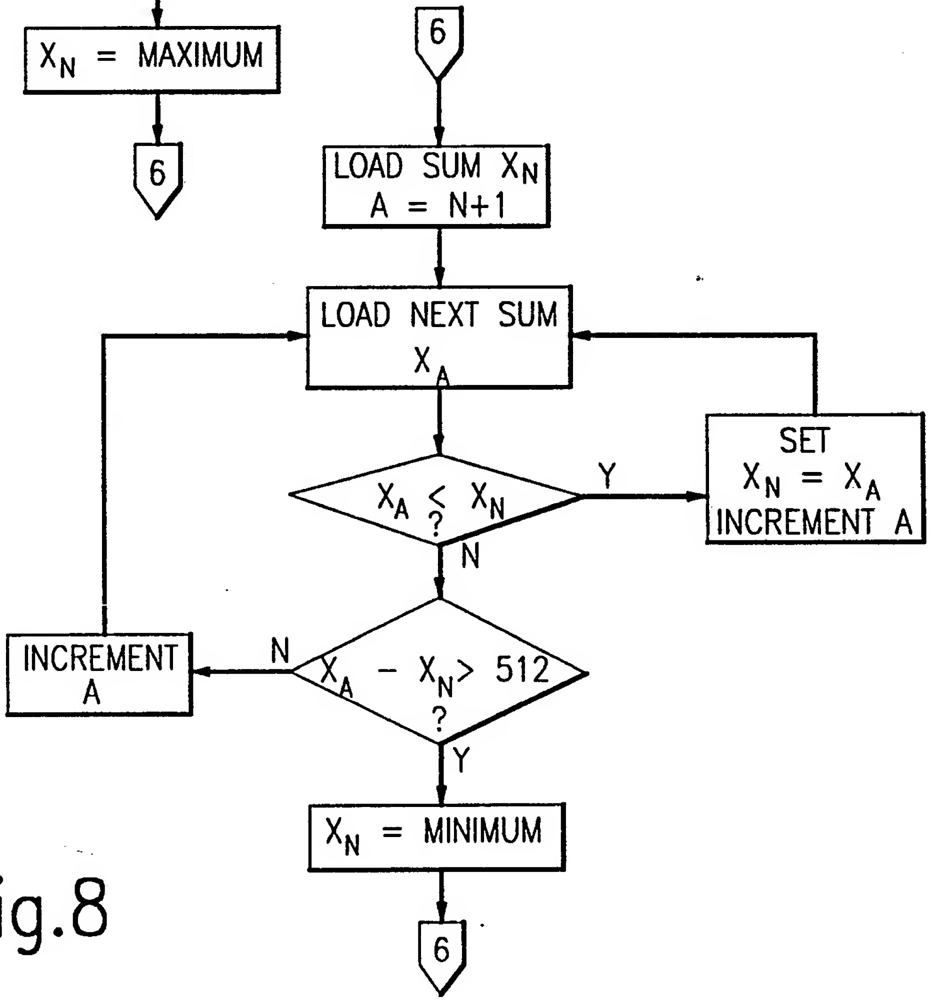

6
LOAD SUM X_N
A = N+1
LOAD NEXT SUM X_A
$X_A < X_N$?
Y
SET $X_N = X_A$ INCREMENT A
N
$X_A - X_N > 512$?
N
INCREMENT A
Y
X_N = MINIMUM
6

Fig.8

INTERNATIONAL SEARCH REPORT

International Application No PCT/US 93/03667

I. CLASSIFICATION OF SUBJECT MATTER (if several classification symbols apply, indicate all)[6]

According to International Patent Classification (IPC) or to both National Classification and IPC

Int.Cl. 5 G10H1/00; G10H1/40

II. FIELDS SEARCHED

Minimum Documentation Searched[7]

Classification System	Classification Symbols
Int.Cl. 5	G10H

Documentation Searched other than Minimum Documentation to the Extent that such Documents are Included in the Fields Searched[8]

III. DOCUMENTS CONSIDERED TO BE RELEVANT[9]

Category °	Citation of Document, [11] with indication, where appropriate, of the relevant passages [12]	Relevant to Claim No.[13]
X	JP,A,64 091 189 (YAMAHA) 10 April 1989 & JP,A,64 091 190 (YAMAHA) 10 April 1989 & US,A,5 159 140 (KIMPARA ET AL.) 27 October 1992 see column 1, line 53 - column 2, line 11 see column 5, line 20 - column 6, line 54; figure 5	1,8,9

X	WO,A,8 402 416 (ETAT FRANCAIS) 21 June 1984 see page 3, line 20 - page 8, line 19; figures 1,2	1,7-9
A		2-6,23, 31,34, 68,78

	-/--	

° Special categories of cited documents : [10]

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

IV. CERTIFICATION

1

Date of the Actual Completion of the International Search: 12 JULY 1993	Date of Mailing of this International Search Report: 16 JUL 1993
International Searching Authority: EUROPEAN PATENT OFFICE	Signature of Authorized Officer: PULLUARD R.J.P.

III. DOCUMENTS CONSIDERED TO BE RELEVANT (CONTINUED FROM THE SECOND SHEET)

Category °	Citation of Document, with indication, where appropriate, of the relevant passages	Relevant to Claim No.
X	WO,A,8 504 065 (VEITCH) 12 September 1985 see page 4, line 21 - page 7, line 27 see page 24 - page 25; figures 1,2	1,2,12, 16,31 34
A		3-10,13, 17-30, 32,33, 35-49, 51-88
A	DE,A,3 643 018 (BERTONCINI) 23 June 1988 see the whole document	1

ANNEX TO THE INTERNATIONAL SEARCH REPORT ON INTERNATIONAL PATENT APPLICATION NO. US 9303667 SA 73723

This annex lists the patent family members relating to the patent documents cited in the above-mentioned international search report.
The members are as contained in the European Patent Office EDP file on 12/07/93
The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

Patent document cited in search report	Publication date	Patent family member(s)		Publication date
JP-A-64091189		None		
WO-A-8402416	21-06-84	FR-A-	2537755	15-06-84
		EP-A,B	0112761	04-07-84
		EP-A-	0142179	22-05-85
		US-A-	4658427	14-04-87
WO-A-8504065	12-09-85	AU-B-	571674	21-04-88
		AU-A-	4115085	24-09-85
		DE-A-	3584448	21-11-91
		EP-A,B	0208681	21-01-87
		EP-A-	0306602	15-03-89
		JP-T-	61502158	25-09-86
		US-A-	4739400	19-04-88
		US-A-	4688090	18-08-87
DE-A-3643018	23-06-88	None		

PUB-NO: WO009322762A1

DOCUMENT-IDENTIFIER: WO 9322762 A1

TITLE: APPARATUS AND METHOD FOR TRACKING MOVEMENT TO GENERATE A CONTROL SIGNAL

PUBN-DATE: November 11, 1993

INVENTOR-INFORMATION:

NAME	COUNTRY
REDMANN, WILLIAM GIBBENS	N/A
PETERSON, MICHAEL HARVEY	N/A

ASSIGNEE-INFORMATION:

NAME	COUNTRY
WALT DISNEY PROD	US

APPL-NO: US09303667

APPL-DATE: April 20, 1993

PRIORITY-DATA: US87435492A (April 24, 1992)

INT-CL (IPC): G10H001/00 , G10H001/40

EUR-CL (EPC): G10H001/00 , G10H001/40

US-CL-CURRENT: 382/107

ABSTRACT:

The invention permits the generation of multipurpose control signals by tracking movement that occurs within a field of view. In particular, a ''Guest Controlled Orchestra'' utilizing these inventive principles permits a layman guest to step into the shoes of an orchestra conductor, and through image processing, conduct the performance of a prerecorded music score. A video camera captures a field of view encompassing the guest for generation of a digital image. The field of view is sampled in left and right windows and the intensity of pixels within the windows are compared with a past image to determine if intensity change exceeds a predetermined threshold. A center of movement is computed for each window by averaging coordinates of each such pixel, and the centers of movement stored for future use. By analyzing change in centers of movement, tempo and volume are derived. Volume is derived from the quantity of pixels that correspond to the predetermined intensity change, and which therefore represent movement. Prerecorded audio data are formatted into MIDI audio commands, and together with video frame advance commands, are processed and output in response to these derived signals.